Evaluation of Incentive-compatible Differentiated Scheduling for Packet-switched Networks

by

Yunfeng Lin

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2005

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Communication applications have diverse network service requirements. For instance, *Voice over IP* (VoIP) demands short end-to-end delay, whereas *File Transfer Protocol* (FTP) benefits more from high throughput than short delay. However, the Internet delivers a uniform best-effort service. As a result, much research has been conducted to enhance the Internet to provide service differentiation. Most of the existing proposals require additional access-control mechanisms, such as admission control and pricing, which are complicated to implement and render these proposals not incrementally deployable. *Incentive-compatible Differentiated Scheduling* (ICDS) provides incentives for applications to choose a service class according to their burst characteristics without additional access-control mechanisms.

This thesis investigates the behaviour of ICDS with different types of traffic by analysis and extensive simulations. The results show some evidences that ICDS can achieve its design goal. In addition, this thesis revises the initial ICDS algorithm to provide fast convergence for TCP traffic.

Acknowledgements

First and foremost, I would like to thank my advisor, Professor Martin Karsten. This thesis would not have been possible without his guidance and patience over the last two years. I am also grateful to Professor Srinivasan Keshav and Professor Paul Ward for taking the time to read my thesis.

Thanks to James She, Aaditeshwar Seth for rewarding discussions in research projects. Suihong Liang, Jun Chen, Sheng Zhang, Zonglin Zhou and many others have helped me much, and raised my morale. James She, Anand Subramanian, and Bassam Aoun share a comfort working environment with me and keep me awake at nights. Nicole Keshav has provided me with a pleasant experience for learning English. Finally, my parents and my sister still love me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Currently, the Internet provides a uniform best-effort service for all applications. "Best effort" means that the Internet transmits packets with no service commitments: packets entering the Internet are neither guaranteed to arrive at their destination, nor are they warranted to arrive within a delay bound. Furthermore, the Internet does not provide different services to packets in different flows. The Internet is highly scalable and simple to implement because of this uniform best-effort characteristic. The routers in the Internet do not need to record flow states and the scheduling algorithms employed in routers, such as FIFO DropTail Queueing, are simple. In addition, the flat-rate type of simple commercial agreements of the Internet are believed to be one of the reasons for its rapid deployment.

The services provided by the Internet to applications have the characteristics in terms of throughput, delay, delay jitter, and loss. Applications have diverse service requirements. For example, *Voice over IP* (VoIP) prefers short end-to-end delay, whereas *File Transfer Protocol* (FTP) is not so sensitive to delay. Much research work has been done to enhance the best-effort service of the Internet to differentiated services and guaranteed services.

Most existing service-differentiation approaches, called *elevated services*, attempt to

provide better services than the best-effort service. Additional mechanisms, such as resource reservation and admission controls or pricing and policing, are usually required to implement elevated services. This introduces several deployment issues. (1) The implementation cost of the additional mechanisms is much higher than the simple best-effort service. (2) The inter-domain pricing model is not clear. (3) Upgrading the Internet from a flat-charge model for the best-effort service to elevated services is not incremental.

Recently, a new approach, called *non-elevated services*, has been developed to provide "different but equal" services [17, 24, 28]. Such services trade delay for loss or throughput; that is, a service class with short delay has high loss or low throughput whereas a service class with long delay has low loss or high throughput. Applications using non-elevated services choose a service class based on their preference. For example, VoIP prefers the service class with short delay and low throughput whereas FTP favours the service class with long delay and high throughput. Non-elevated services do not require additional access-control mechanisms or price differentiation because of this "equal" nature. Consequently, there is no implementation overhead for the additional control mechanisms. Moreover, upgrading from the best-effort service to non-elevated services can be incremental.

Incentive-compatible Differentiated Scheduling (ICDS) is a new member of the family of non-elevated services in the sense that it can be used without additional access-control mechanisms. It is based on the observation that bursty traffic requires more buffer than smooth traffic to achieve the same loss rate. ICDS provides services with different delays. A service with a short delay usually means less buffer. Consequently, applications with smooth traffic and a requirement for short end-to-end delay, such as VoIP, have an incentive to choose the short-delay service without losing too many packets; applications with bursty traffic and a preference of high throughput, such as FTP, are more likely to choose the long-delay service to avoid a high loss rate. Furthermore, ICDS can be used as a building

block with admission control to provide delay-guaranteed services.

The mechanism of ICDS is simple. It allocates the link bandwidth in proportion to the arrival rates of the traffic in all service classes in order to provide non-elevated services from the throughput perspective. The basic idea and a prototype on the Network Simulator [3] of ICDS are developed by Martin Karsten. The goal of this thesis is to study the behaviour of ICDS and determine whether ICDS achieves its design goal. The contributions of this thesis are summarized as follows:

1. This thesis investigates the behaviour of ICDS by analysis and simulations under different types of traffic. In particular, it provides a model to describe the behaviour of TCP traffic with ICDS based on an existing model on the behaviour of TCP traffic with DropTail queueing.

2. The examination of the convergence of TCP traffic in ICDS leads to a clear understanding of the effects of the minimal-rate mechanism and the add-rate option.

3. The investigation in this thesis discovers that it is important to adjust service rates at the same frequency for all the classes in ICDS to achieve a fast convergence for TCP traffic.

4. This thesis also provides numerical analysis for the bounds of the errors introduced by the technique to remove division operations, and a simple proof of the decay property of the *Efficient Time Sliding Window* (ETSW) algorithm.

The remainder of this thesis starts with an introduction to the background and related work of ICDS in Chapter 2. Chapter 3 describes the design and implementation of ICDS. Chapter 4 presents the analysis and simulation results. Chapter 5 concludes this thesis and suggests future work.

Chapter 2

Related Work and Background

This chapter reviews the background and related work for *Incentive-compatible Differentiated Scheduling* (ICDS). Section 2.1 introduces the delay requirements of applications. Section 2.2 and 2.3 review the related work. Section 2.4 presents the background.

2.1 Application Delay Requirements

The delay requirements of applications are not identical. Typically, interactive applications require shorter delay than non-interactive applications. *Elastic* applications are more tolerant to delay than *real-time* applications [8]. Simply speaking, real-time applications expect packets to arrive before a certain deadline. If the packets arrive later than expected, they are useless. On the contrary, elastic applications usually can wait for packets. These two types of applications are discussed in more detail in the following sections. Figure 2.1 is a summary of the delay requirements of certain applications.

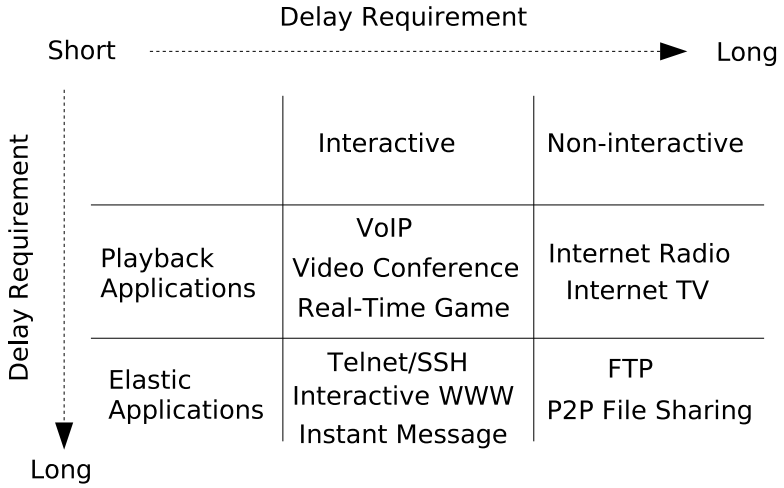


Figure 2.1: Application delay requirements.

2.1.1 Playback Applications

A major type of real-time applications are called *playback* applications. They replay the signals from senders at receivers. For example, VoIP replays the speaker's voice to the listener; Internet Radio replays the music or the voice of the anchorman to the audience.

A playback application works as follows. The sender encapsulates the encoded digitalized signal in packets and transmits them through the Internet to the receiver. When the packets arrive at the receiver, it decodes the signal from the packets and replays it. To restore the signal to the receiver at the same rate as it is encoded at the sender, playback applications usually put the *playback time* into the packets to indicate the receiver when to play the contents of the packets.

A *playback delay* exists between the time when the signal is encoded at the sender and when it is played back at the receiver. Clearly, if the transmission delay of all the packets is identical, the playback time carried in them is set at a value such that the playback delay is the transmission delay. However, the delays of all the packets are not identical in a packet-

switched network where queueing delays change over time. In playback applications, if the packets arrive early, their contents are useful to construct the signal; they are stored in a buffer and replayed later at the playback time. If the packets arrive late, their contents are useless and they are discarded. Usually, playback applications can tolerant some degree of packet drops by interpolating the signal in the lost packets from their neighbouring packets.

Although playback applications expect packets to arrive before the playback time, they are usually tolerant to some variation of delay. Rigid playback applications estimate a fixed playback delay by prior knowledge. However, most modern playback applications detect the appropriate playback delay bound by monitoring the percentage of the packets that arrive late and need to be discarded. If there are too many of dropped packets, the quality of the applications degrades and the playback applications increase the estimated playback delay.

2.1.2 Elastic Applications

Elastic applications are the traditional data applications supported in the best-effort Internet. In contrast to real-time applications, elastic applications do not expect that packet delays are within a certain range. Nevertheless, this does not mean that packet delays are irrelevant to the performance of elastic applications. Their performance does degrade when packet delays are long.

Essentially, elastic applications differ from real-time applications at the time to process packets at receivers. Elastic applications process the packets immediately when they arrive whereas real-time playback applications store them in a buffer, and replay their contents later at the playback time. Furthermore, elastic applications do not usually discard packets once they arrive at receivers. Although elastic applications are not very sensitive to packet

delays, they also have a delay preference depending on whether they are interactive or non-interactive.

The values of the delay requirements of a few applications are given as follows. The quality of VoIP (an interactive real-time application) becomes annoying when its end-to-end delay exceeds 150ms [32]. The end-to-end delay consists of network delay and processing delay (packetization, encoding and decoding delay). Therefore, the required network delay is less than 150ms. Figure 2.2 illustrates the delay requirements of some popular applications (most of them are elastic TCP applications).

Currently, the Internet offers only a uniform best-effort service which is not sufficient to accommodate the diverse service requirements of applications. Service differentiation has been an active research topic for more than a decade. The existing approaches can mainly be classified as elevated services and non-elevated services. *Elevated services* provide service classes better than the best-effort service. To control the use of privileged service classes, the implementation of elevated services requires control-plane mechanisms (e.g., resource reservation, charging, and policing) in addition to the necessary upgrades for data-plane mechanisms (packet schedulers employed in the router data-forwarding path). Elevated services are difficult to deploy because of the high implementation complexity of the control-plane mechanisms and non-incrementally deployable property (i.e., the control-plane mechanisms usually need to be deployed everywhere to operate or avoid denial-of-service attacks [40]). *Non-elevated services* provide "different but equal" service classes. Therefore, no control-plane mechanisms are required for implementation and the deployment can be incremental. The representative approaches in both categories are now reviewed.

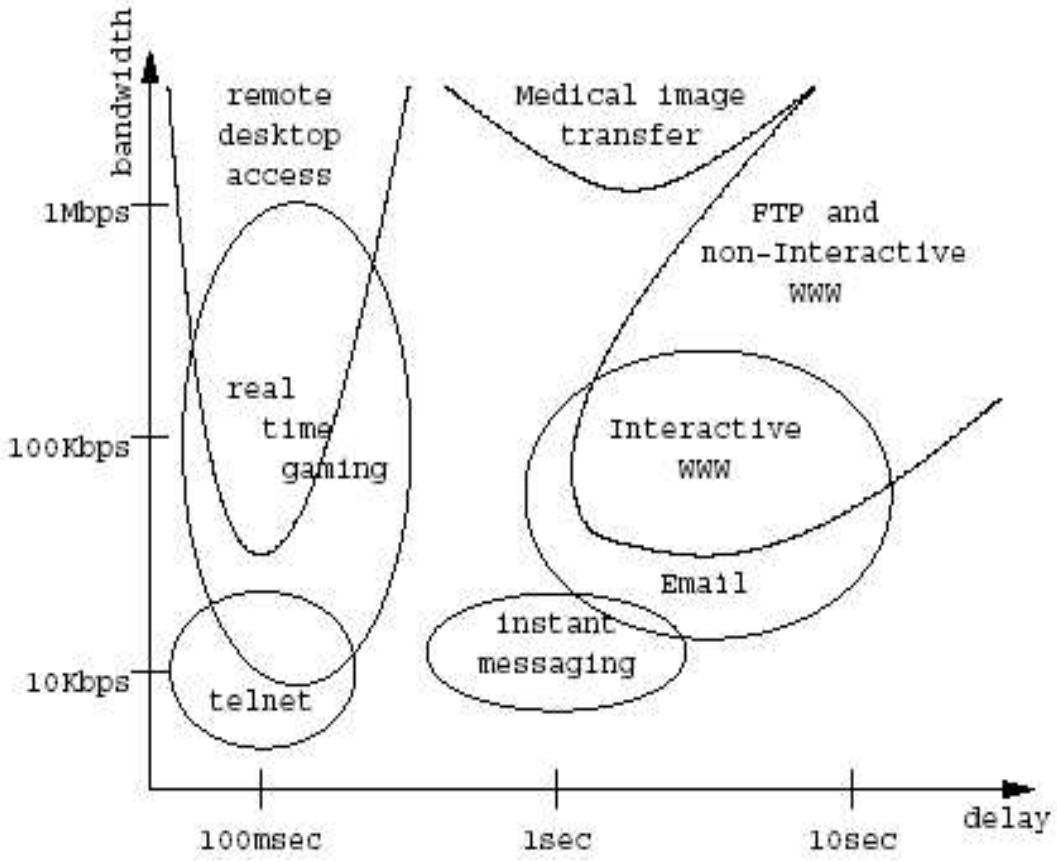


Figure 2.2: Delay and bandwidth requirements of some popular applications. Courtesy of Noureddine and Tobagi [35].

2.2 Elevated Services

Elevated services include resource-reservation models and priority models. The following describes the two representative approaches discussed in the IETF working groups.

2.2.1 Integrated Services

Integrated Services (IntServ) [8] are a resource-reservation model. IntServ provides end-to-end service guarantees (e.g., end-to-end delay and bandwidth) to individual flows via resource reservation and admission controls. IntServ is not scalable because core routers must maintain per-flow state which imposes large computational costs. IntServ is not deployed currently, possibly due to the limited scalability, the implementation complexity of the control-plane mechanisms such as the signalling protocol, management, accounting, and the all-or-nothing upgrade from the best-effort Internet.

2.2.2 Differentiated Services

The basic design of *Differentiated Services* (DiffServ) [7] is a priority approach. It provides per-hop service differentiation to aggregates of flows (called *classes*). Therefore, DiffServ is more scalable than IntServ. In the DiffServ architecture, packets are marked with class identifiers at the border routers of a DiffServ domain. After packets enter the domain, core routers forward them with the service rules associated with their class identifiers. Two per-hop services have been standardized in the DiffServ architecture: *Premium service* [11] and *Assured service* [26]. Premium packets experience low queueing delay and low loss rate in the Premium service. Packets in different classes have different drop rates in the Assured service. DiffServ provides better services to some classes than others. It controls the resource allocation of the privilege classes by charging their users more.

DiffServ can also be used to provide service guarantees. For example, the QBone Premium Service (QPS) [4] offers a virtual leased-line service model by the Expedited Forwarding [11] per-hop service.

However, charging and access control are complex to be implemented and render DiffServ not incrementally deployable. QPS is suspended and may never be continued because the cost for deployment is higher than the benefits [40]. The top reasons are "poor incremental deployment properties, intimidating new complexity for network operators, missing functionality on routers, and serious economic challenges" [40].

2.3 Non-elevated Services

Non-elevated services provide "different but equal" services; that is, they provide services that are a trade-off between delay and loss (or between delay and throughput). Because different service classes are "equal", control-plane mechanisms such as price differentiation and admission control are not required. Therefore, non-elevated services can be incrementally deployed from the best-effort service of the Internet. Furthermore, these services retain the best effort, flat-rate type of commercial agreements which are believed to be one of the reasons for the rapid deployment of the Internet. This section is organized as follows. Sections 2.3.1, 2.3.2, and 2.3.3 describe the three existing proposals. Section 2.3.4 compares them and discusses their potential weaknesses.

2.3.1 Best Effort Differentiated Services

Best Effort Differentiated Services (BEDS) [17] trades delay for loss. It provides two service classes. The service class for UDP traffic (the traffic of delay-sensitive applications) has a short delay and a high loss rate whereas the service class provided for TCP traffic (the

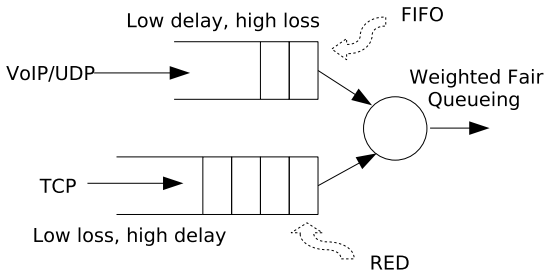


Figure 2.3: Architecture of BEDS.

traffic of throughput-sensitive application) has a long delay and a low loss rate. BEDS enforces the ratio of the delay and loss of TCP traffic to UDP traffic. The delay ratio and the loss ratio are two configuration parameters.

Figure 2.3 illustrates the architecture of BEDS. It puts TCP traffic into the *Random Early Detection* (RED)[22] queue and UDP traffic into the FIFO DropTail queue. BEDS employs *Weighted Fair Queueing* (WFQ) [12] to dynamically adjust the ratio of the service rate of the TCP queue to the UDP queue.

BEDS uses the *Backlog-Proportional Rate* [14] algorithm to maintain a fixed ratio of queueing delays; that is, BEDS adjusts the weight of the TCP queue w_{TCP} and the weight of the UDP queue w_{UDP} in the WFQ scheduler as follows:

$$\frac{w_{UDP}}{w_{TCP}} = \delta * \frac{q_{UDP}}{q_{TCP}}, \tag{2.1}$$

where q_{UDP} and q_{TCP} are the current queueing delays of the TCP queue and the UDP queue respectively. δ is the delay ratio.

The drop rate p_{TCP} of the TCP traffic is calculated by a RED control function [16] with the characteristic of a low drop rate and a long delay. The drop rate p_{UDP} of UDP

traffic is determined by

$$p_{UDP} = \lambda p_{TCP}, \tag{2.2}$$

where λ is the drop-rate ratio.

If the UDP traffic is TCP-friendly and complies to TCP Friendly Rate Control (TFRC) [20], it has the same response function as TCP traffic. The average throughput $T(p, R)$ is a function of the drop rate p and the round trip time R as expressed follows [36]:

$$T(p, R) = \frac{M}{R}\sqrt{\frac{3}{2bp}} + o(\frac{1}{\sqrt{p}}) \approx C\frac{1}{R\sqrt{p}}, \tag{2.3}$$

where M is the average packet size and b is a constant. The throughput ratio of the TCP traffic and the TFRC traffic can be controlled by the delay ratio and the loss ratio. For example, given (2.1), (2.2), and (2.3), the average throughput of the TCP traffic and the TFRC traffic is equal if the delay ratio δ and loss ratio λ are set as $\delta = \sqrt{\lambda}$ (assuming that the propagation delay is short enough to be neglected such that the ratio of the queueing delays approximately equals to the ratio of the round trip times).

2.3.2 Equivalent Differentiated Services

Equivalent Differentiated Services (EDS) [24] trades delay for loss similar to BEDS. EDS provides service classes either with a short delay and a high loss rate or a long delay and a low loss rate as shown in Figure 2.4. The delay ratios and loss ratios of the multiple classes are configurable. EDS is implemented by the Waiting-Time Priority scheduler [14] (a proportional-delay scheduler) and the Proportional Loss Dropper [13].

In EDS, delay-sensitive applications adaptively choose the service class that meets the delay requirement and has the minimal possible loss rate by monitoring the end-to-end delay. Assume EDS provides N service classes with delays $d_i < d_j$ in ascending order and

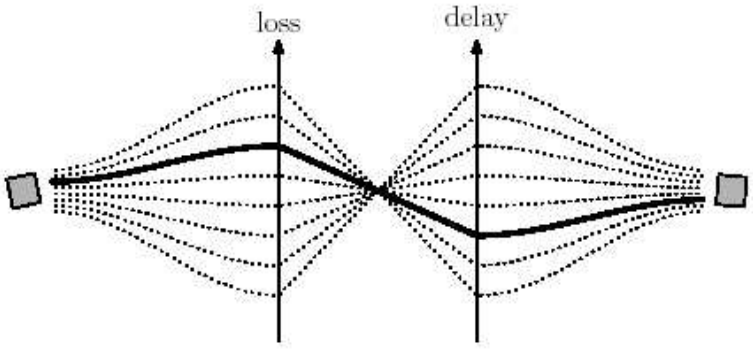


Figure 2.4: EDS model with eight classes. The thick line is a class with a loss rate higher than the average loss rate and a lower delay than the average delay. Courtesy of Gaidioz and Primet [24].

loss rates $p_i > p_j$ in descending order, where $1 \leq i < j \leq N$. Then the service class for an application with the delay requirement of D is class k with $k = \max(j)$, where $d_j < D$.

Although the implementation of EDS and BEDS differs, their service models are similar. Both BEDS and EDS deliver proportional delay differentiation and proportional loss differentiation. Therefore, like BEDS, EDS can adjust the delay ratio and the loss ratio to control the throughput ratio between service classes if the traffic of different classes is TCP-friendly.

2.3.3 Alternative Best Effort

Alternative Best Effort (ABE) [28] provides two service classes. One is a fixed short-delay service class for delay-sensitive applications, called *green* service; the other is a service class with no delay guarantee for throughput-sensitive applications, called *blue* service. ABE trades delay for throughput. Green service provides fixed short delay and low throughput whereas blue service has long delay and high throughput.

Green traffic does not hurt blue traffic in ABE; that is, ABE provides blue traffic *local transparency* and *throughput transparency*. Local transparency guarantees that the blue packets experience the same or shorter delay in ABE than the delay they would experience when all traffic is served by a best-effort service (e.g., FIFO DropTail queueing). If a blue packet is not dropped in the best-effort service, it is not dropped in ABE. Throughput transparency ensures that the throughput of blue traffic is not less than the throughput of green traffic when both of them are TCP-friendly.

ABE is implemented by the packet scheduler, *Duplicate Scheduling with Deadlines* (DSD), which is a variant of Earliest Deadline First schedulers [44]. Figure 2.5 illustrates DSD. The green and blue packets are enqueued in two queues. A virtual queue (e.g., FIFO DropTail queueing) implementing the flat best-effort service is used to drop packets when the buffer is full and tag timestamps for the blue packets. Each arrival packet is virtually duplicated. The duplicate is enqueued into the virtual queue.

A blue packet is dropped if its duplicate is dropped in the virtual queue. Otherwise, it is accepted into the blue queue. The timestamps of the blue packet is $t + vd$, where t is the current time and vd is the queueing delay of its duplicate in the virtual queue. A green packet and its duplicate are admitted into the green queue and the virtual queue respectively, if the green packet passes the *green acceptance test*, which accesses whether the green packet can be transmitted with a queueing delay shorter than d as illustrated in Figure 2.5. If assuming at time t that the queueing delay of the green queue is l_g and the time to transmit the blue packets with deadlines less than $t + d$ is l_b, a green packet is accepted into the green queue if $d > l_g + l_b$. The deadline of a green packet is $t + d$. DSD guarantees the delay bound of the green packets and the local-transparency property for the blue packets by serving the green packets and the blue packets within their deadlines.

The response function of TCP, Equation (2.3) [36] on page 12, indicates that the average

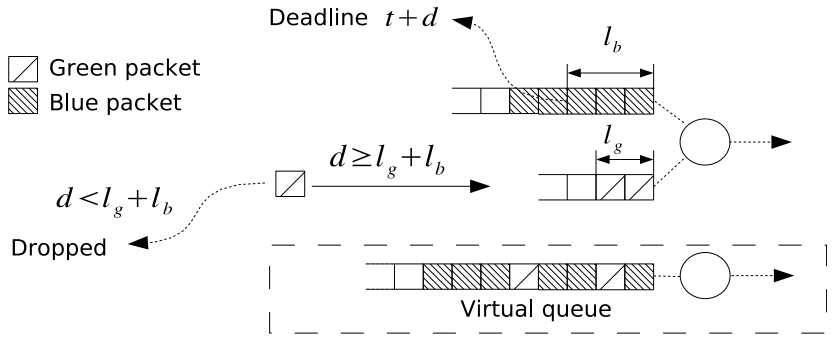


Figure 2.5: ABE and the green acceptance test.

throughput of TCP is approximately inverse proportional to the round trip time. If the green traffic is TCP-friendly with the same response function as TCP, then the green packets with a shorter queueing delay (thus a small round trip time) would get higher throughput than the blue packets, if there are no other mechanisms. This is not desired because the green service is better than the blue service. ABE therefore enforces that the throughput of the blue traffic to be larger than the green traffic (i.e., the throughput transparency for the blue traffic). ABE monitors the delay and loss ratio of green to blue traffic and uses a feed-back controller to dynamically adjust the probability g to serve the green packet, when both the green packet and the blue packet at the head of queue can wait (their deadlines are both larger than the current time). Clearly, if g is large, green packets are favoured and vice versa. If ABE does not always serve a green packet ($g < 1$), some green packets can violate their deadline even though they pass the green acceptance test. Therefore, ABE has to search and drop all the stale green packets in the dequeue operation. The feed-back controller adaptively adjusts g to a reasonable small value to increase the drop rate of green traffic. Consequently, the throughput of the green traffic suffers and can be less than the blue traffic.

2.3.4 Discussion

The existing approaches of non-elevated services fall into two categories. BEDS and EDS are proportional delay and loss models whereas ABE is an absolute delay model. Two advantages make ABE more attractive. The maximal queueing delay experienced by packets varies when traffic load changes in BEDS and EDS. Therefore, applications must adopt an adaptive mechanism to choose the service class that meets the end-to-end delay requirement as proposed in EDS. ABE does not require this additional mechanism. The second advantage of ABE is that it needs to configure only an absolute delay which is relatively clear given the application delay requirements. In contrast, BEDS and EDS need to configure delay ratios and loss ratios. Reasonable delay ratios and loss ratios are not so obvious.

However, ABE has its own weaknesses. It provides only two delay classes which are not flexible. As shown in Figure 2.2 on page 8, the delay requirements of applications are more diverse than two values. Furthermore, ABE has high implementation complexity. The worst-case complexity of searching the stale green packets in the dequeue operation of ABE is in proportion to the length of the green queue. The worst-case complexity of the green acceptance test is in proportion to the length of the blue queue.

Another argument is about ABE's definition of throughput transparency. ABE enforces the aggregate throughput of the blue traffic to be always larger than the green traffic at any time, ignoring the number of flows within them. This may not be reasonable. For example, 1000 TFRC flows and 1 TCP flow compete for a bottleneck link. If the throughput of the TCP traffic is enforced to be larger than the throughput of the TFRC traffic, the performance of an individual TFRC flow is significantly worse than the case when all the traffic is served by the best-effort service. A more reasonable definition of throughput transparency may be that the throughput of an individual TCP flow is larger than an

individual TFRC flow. In such a case, the complicated feed-back controller to adaptively adjust the probability g to transfer green packets would not be necessary. A fixed g which renders the throughput of one TCP flow larger than the throughput of one TFRC flow may be sufficient.

Dr. Martin Karsten proposes a new service-differentiation approach: *Incentive-compatible Differentiated Scheduling* (ICDS). ICDS provides absolute delay services similar to ABE while addressing its deficiencies. ICDS provides multiple delay classes and its implementation is more efficient than ABE. ICDS (or a generalized ABE which provides multiple delay service classes) can be used either as a type of non-elevated services or as a building block combined with admission control mechanisms to provide guaranteed delay services.

The existing approaches of non-elevated services provides incentives for applications to choose their service class according to their delay and throughput preference. Delay-sensitive applications prefer a short-delay service with some losing in throughput, whereas throughput-sensitive applications favour a high-throughput service with a relative long delay. It is argued that ICDS provides a new kind of incentive for applications to choose a service class in accordance with the burst characteristics of their traffic. A common observation is that bursty traffic needs more buffer, whereas smooth traffic needs less buffer to attain the same loss rate. More buffer normally means a larger queueing delay and vice versa. Therefore, applications with smooth traffic have no disincentive to choose the short-delay service without the fear of losing too many packets, whereas applications with bursty traffic have an incentive to choose the long-delay service to avoid a high loss rate.

2.4 Background

ICDS adopts a rate scheduler, a packet version of the Generalized Processor Sharing (GPS) [12, 37]. The evaluation of ICDS with TCP traffic is principally related to the research on TCP traffic with the size of router buffers. The following two sections briefly introduce these background material of ICDS.

2.4.1 Generalized Processor Sharing

The *Generalized Processor Sharing* (GPS) algorithm is a generalization of the uniform processor sharing algorithm [31] to share a service among multiple users. The work presented in [37] studies the properties of GPS in the context of link-bandwidth sharing. A link with a speed r is served by GPS. Session i is assigned a weight ϕ_i. A busy session i (with packets waiting for transmission) is guaranteed to be served with the rate $r_i = \frac{r\phi_i}{\Sigma_{j \in B(t)}\phi_j}$, the fraction of the link speed determined by its relative weight among all the busy sessions $B(t)$.

GPS is an idealized fluid model with the following assumptions: traffic can be divided infinitely; the link can serve multiple sessions simultaneously. However, in a packet network, the minimal traffic unit is one packet; a link can transmit one packet from only one session at a time. Much research work has been done to emulate GPS with a packet algorithm. The performance of a packet GPS algorithm is evaluated by accuracy and complexity. The state-of-the-art packet GPS algorithm is L-WF^{2}Q [41] with $O(1)$ deviation from GPS in terms of packet times and $O(\log(n))$ complexity in terms of the number of sessions served.

2.4.2 Router Buffer Size and TCP Traffic

Internet routers contain buffers to absorb the temporary imbalances between the traffic load and the link capacity. The optimal size of a router buffer is not obvious. A large buffer size potentially generates a long queueing delay whereas a small buffer size can cause too many packet losses. Because the major type of traffic in the Internet is TCP traffic, the buffer size of a router is mainly related to the behaviour of TCP traffic. TCP traffic is generated by both long TCP flows (with a large amount of data such as FTP traffic) and short TCP flows (with limited data such as interactive web traffic). The measurements on commercial networks [23] suggest that over 90% of the traffic is from long TCP flows in the Internet. Therefore, router buffer sizes are principally determined by long TCP flows [6]. Two performance characteristics are related to the size of a router buffer under TCP traffic: the utilization of the bottleneck link and the fairness among TCP flows. The following is presented with the assumption that there is only one bottleneck router in a data path.

First, the buffer size of the bottleneck router should be large enough to fully utilize the bottleneck link. In the scenario where a single TCP flow transmits data on a bottleneck link, if a packet is lost, TCP's sending rate is roughly reduced by half through the congestion avoidance algorithm [5] such that the sending rate is less than the link capacity. During the following period until the sending rate climbs back to the link bandwidth, only the data in the buffer are available to be transmitted. A router needs a buffer with a size B at least the delay-bandwidth product ($B = \overline{RTT} \times C$, where $\overline{RTT}$ is the average end-to-end two-way propagation delay and C is the link capacity) to hold sufficient data to keep the link busy [6, 42]. It turns out that a small number of TCP flows (synchronization is common when the number of flows is fewer than 100 [6]) synchronize and exhibit the same behaviour as a single TCP flow such that they also need a buffer with a size at lease the delay-bandwidth

product to fully utilize the link.

If a large number of TCP flows share a bottleneck link, the size of the router buffer B should be larger than $(\overline{RTT} \times C)/\sqrt{n}$ (n is the number of TCP flows) to fully utilize the bottleneck link [6]; that is, the required router buffer size decreases when the number of TCP flows grows. TCP flows desynchronize if the number of TCP flows is large [6] (in-phase synchronization is very rare above 500 concurrent flows [6]). So the the congestion window sizes of different TCP flows are independent. The probability distribution of the sum of all the congestion window sizes is a normal distribution and its standard deviation is $1/\sqrt{n}$ of the standard deviation of the probability distribution of one individual congestion window size by the central limit theorem. As a result, the aggregate traffic of a large number of TCP flows is less bursty than a small number of synchronized TCP flows. Consequently, the required size of the router buffer decreases.

The size of the router buffer should be large enough to maintain the fairness among multiple TCP flows [34, 38]. TCP's throughput is adversely affected if fast retransmit [5] cannot detect most of the packet losses. Fast retransmit requires that the size of the congestion window of a TCP flow is larger than three in terms of the number of packets. If the average size of congestion windows is less than three, some TCP flows are transmitting data smoothly with most packet losses detected by fast retransmit whereas others are always idle in the TCP retransmit timeouts [38]. Hence, the size of a router buffer should be large enough such that the average size of congestion window of all TCP flows is larger than three.

The sum $(B + \overline{RTT} \times C)$ of the size of the bottleneck router buffer B and the delay-bandwidth product $\overline{RTT} \times C$ limits the amount of data (i.e., the sum of the congestion window sizes of all TCP flows) that can be injected into the network. Therefore, the average congestion window size w is $(B+\overline{RTT}\times C)/n$, where n is the number of TCP flows. Because

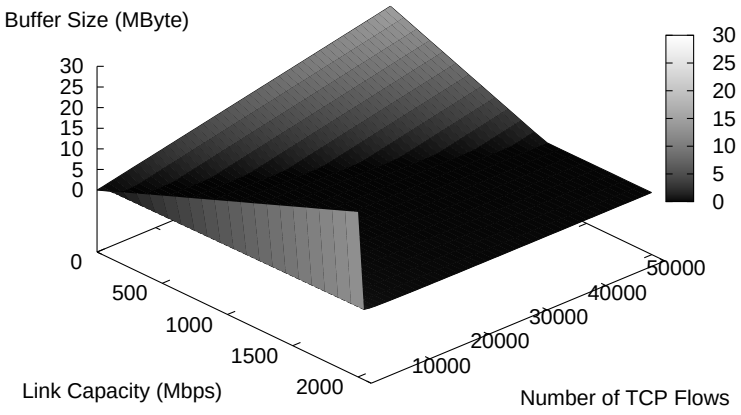


Figure 2.6: TCP buffer requirement combining the impact of the link utilization requirement and fairness. The round trip propagation delay is 100ms.

of the fairness requirement of $w \geq 3MSS$, where MSS is the size of a segment, the size B of the router buffer should satisfy the following relation: $B = nw * MSS - \overline{RTT} \times C \geq 3n * MSS - \overline{RTT} \times C$.

Figure 2.6 shows the minimal acceptable buffer size of a router combining the impact of the link utilization and fairness among flows. The required buffer size increases in two scenarios: (1) When the link capacity is large and the number of flow is small (i.e., the aggregate traffic is bursty), a large buffer is needed to hold sufficient data to fully utilize the link bandwidth; (2) When the link capacity is small and the number of TCP flows is large, a large buffer is required to guarantee the fairness among flows.

Chapter 3

Incentive-compatible Differentiated Scheduling

Dr. Martin Karsten contributes to a large part of this chapter. His work is included here because it is the background and base of this thesis. The contribution from this thesis in this chapter is stated as follows. As presented in Section 3.1.2, this thesis proves that the *Efficient Time Sliding Window* (ETSW) algorithm has the same long-term decay property as the *Time Sliding Window* (TSW) algorithm. As shown in Section 3.1.3, this thesis discovers that ICDS must adjust at the same frequency. In presented in Section 3.4, this thesis analyzes the error bound of the technique to remove the division operation. Finally, this thesis also provides the clear understanding of the functionality of the minimal-rate mechanism and the add-rate option as illustrated in Section 3.5.

This chapter presents the design and implementation of *Incentive-compatible Differentiated Scheduling* (ICDS). ICDS provides multiple service classes with different delays. The service class with a long delay normally means a larger buffer. A common observation is that smooth traffic needs a smaller buffer than bursty traffic to achieve the same loss

rate. ICDS provides incentives for applications to choose a service class according to their burst characteristics. The applications with smooth traffic have no disincentive to choose a short-delay service without the fear of losing too many packets, whereas the applications with bursty traffic have an incentive to choose a long-delay service to avoid a high loss rate. In order to provide non-elevated services from the throughput perspective, ICDS adjusts the service rates of the applications in different service classes in proportion to their sending rates.

Figure 3.1 illustrates the conceptual architecture of ICDS. It separates the traffic with different delay requirements into different queues. The traffic with the same delay requirement and entering the same queue is called a *traffic class* or simply a *class* in the remainder of this thesis. The maximal queueing delay (called *delay target*) of a class is configurable. ICDS measures the arrival rates of all the classes. It uses a packet GPS scheduler to allocate the link bandwidth in proportion to the arrival rates of traffic classes and estimates the queueing delay of a packet by dividing the queue length by the service rate. If the queueing delay of the packet does not exceed the delay target, it is admitted. Otherwise, it is dropped.

One of the design goals of ICDS is efficiency. Because ICDS uses a component of a packet GPS scheduler, it is at least as complex as the packet GPS algorithm employed. The complexity of ICDS is $O(1)$ if the complexity of the packet GPS algorithm is excluded.

This chapter is organized as follows. Section 3.1 presents the design of the arrival-rate estimator. Section 3.2 illustrates the time lag between the arrival-rate estimation and the service-rate adjustment. Section 3.3 introduces the mechanism for strict delay target guarantees. Section 3.4 describes the technique to remove the division operation in the calculation of the queueing delays of packets. Section 3.5 presents the techniques to guarantee and accelerate the convergence of TCP traffic with ICDS.

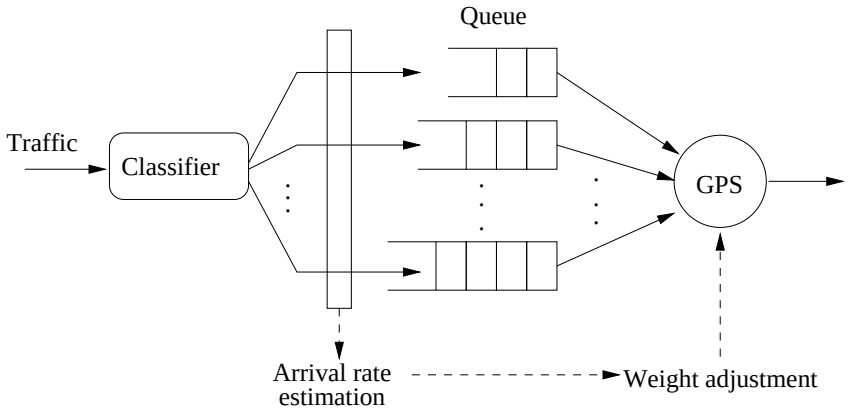


Figure 3.1: Architecture of ICDS.

3.1 Arrival-Rate Estimation

ICDS estimates the arrival rates of traffic classes and use them to determine the service-rate allocation. The value of the service rate is used to calculate the queueing delay of a packet by dividing the queueing length by the service rate. Because the packet GPS scheduler allocates an amount of link bandwidth to a class as the fraction of its weight (its arrival rate) among the total weights (the total arrival rate), the value of the service rate is actually results of the relative arrival rate of this class multiplying with the link capacity. Therefore, ICDS needs to get the relative arrival rates of all the classes. Section 3.3.1 describes the approach used in ICDS.

The second issue with an arrival-rate estimator is that it should smooth out the instantaneous bursts of traffic to avoid the oscillation of the system on a small time scale while provide a fast response when the arrival rate changes on a medium or large time scale. Section 3.3.2 presents the smoothing algorithm adopted in the arrival-rate estimator of ICDS. A by-product of this smoothing algorithm removes the division operation in the rate estimation.

Finally, the update frequencies of the rate estimators of all the classes should be identical and independent of their arrival rates to be *fair*. Section 3.3.3 addresses this issue.

3.1.1 Relative Arrival-Rate Estimation

The most straightforward way to acquire the relative arrival rates is to estimate the absolute arrival rates of all the classes at the same time, and calculate the relative arrival rate of a class by dividing its absolute arrival rate by the total absolute arrival rate of all the classes. This operation must be done periodically to adapt to the change of the traffic load. Such a method may be expensive because it requires timer support and division operations which are significantly slower than addition and multiplication operations [39]. Furthermore, the timer handler has $O(n)$ complexity in terms of the number of traffic classes.

Although timer support may not be very expensive and the timer handler is not directly on the critical path (per-packet processing), ICDS uses an alternative approach to avoid the $O(n)$ complexity and divisions in the first place. ICDS performs the computation of the arrival-rate estimation in per-packet processing. Each class estimates only its relative arrival rate independent of other classes. Therefore, although the total computation cost of rate estimation is not reduced, ICDS distributes the $O(n)$ computation cost in the timer handler to the per-packet processing of each class. The complexity of the arrival-rate estimation is $O(1)$ per packet independent of the number of classes.

ICDS estimates the relative arrival rate directly instead of estimating the absolute arrival rate of each class and calculating the relative arrival rates based on it. The average relative arrival rate r of a class in a time interval is calculated by dividing the amount of the arrival data l by the total arrival data s of all the classes in this time interval which is

expressed as

$$r = \frac{l}{s}. \tag{3.1}$$

A division operation exists here. However, this division operation is replaced by an efficient shift operation in the version of algorithm with smoothing as shown in Section 3.1.2.

Because ICDS calculates the average arrival rate in per-packet processing, naturally, the time interval for calculating the average relative arrival rate can be the time period between two continuous receptions of packets in this class. Clearly, the raw instant rate in such a small time interval is very unstable. Section 3.1.2 addresses this problem by applying a smoothing algorithm on the relative arrival-rate estimation.

Figure 3.2 shows an example to illustrate this idea. The arrival rates of the three classes are identical and constant. Obviously, the relative arrival rates of them are all 1/3. All packets have the same size. The packet from class 2 follows the packet from class 1; the packet from class 3 follows the packet from class 2. Equation (3.1) calculates the relative arrival rate of class 1 between the receptions of the two packets in class 1 by $l/s = l/(3 * l) = 1/3$, which is the expected correct value.

If all traffic classes transmit data in a constant rate, Equation (3.1) obtains the precise relative arrival rate for all the classes. The sum of all the relative arrival rates is 1. However, if some classes transmit data at a varying rate, Equation (3.1) gets the average relative arrival rate in a time interval. The sum of all the relative arrival rates may not be 1 because the time intervals of the arrival-rate estimations in different classes are not identical.

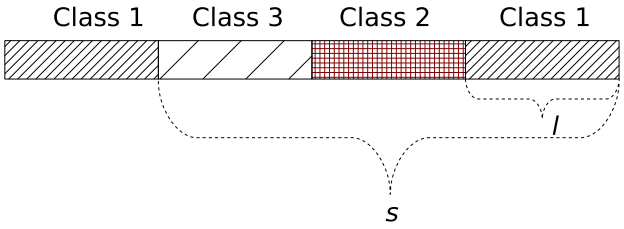


Figure 3.2: Relative rate estimation. Each rectangle represents a packet. The relative arrival rate is calculated by l/s.

3.1.2 Smoothing Algorithm in Arrival-Rate Estimation

This section presents the smoothing algorithm adopted in ICDS to smooth out the instantaneous burst of traffic on a small time scale to avoid the instability of the system. In addition, the smoothing algorithm must estimate the average relative arrival rate of each class roughly in the same time interval both in the position and the length, because ICDS uses the estimated arrival rates to adjust the service-rate allocation. This section investigates two common smoothing algorithms: weight-based estimations and time-based estimations. The distinction between them is that the correlation between the estimated moving average and the rate history is dependent on the traffic rate in weight-based estimations, whereas the correlation is independent on the traffic rate in time-based estimations. Therefore, essentially, for the classes with different arrival rates, weight-based estimators calculate arrival rates in the time intervals with different lengths, whereas time-based estimators estimates arrival rates in the time intervals with the same length. They are discussed in more detail in the following.

The smoothing algorithm of the rate estimator is discussed on absolute rates in this section. First, weight-based estimators are presented. The representative approach, the *Exponentially Weighted Moving Average* (EWMA) algorithm measures the moving average

rate R_i when the ith packet arrives:

$$R_i = \alpha R_{i-1} + (1 - \alpha) r_i, \tag{3.2}$$

where r_i is the current instantaneous rate measured when the ith packet arrives; α ($0 < \alpha \leq 1$) is a constant representing the depth of EWMA's memory. r_i is calculated by l_i/δ_i where l_i is the size of the ith arrival packet and δ_i is the time between the $(i-1)$th and the ith packet.

The decay of the estimated rates depends on the packet arrival rates in weight-based estimators because the estimate rate is updated for each packet. If the traffic source sends data fast, the decaying of the rate estimation is also fast, and vice versa. Consequently, the weight-based estimator of a high-speed source forgets the rate history more quickly than a low-speed source.

ICDS allocates service rates based on the estimated arrival rates. The moving averages of arrival rates in different traffic classes need to be in the same time window. Therefore, it is important to guarantee that the decaying speeds of the rate estimators in different classes are identical and independent of the arrival rates.

The second approach, the *Time Sliding Window* (TSW) algorithm [9] decays the rate history over time independent of the arrival rates. The average rate R_i when packet i arrives in TSW is calculated by

$$R_i = \frac{R_{i-1} W + l_i}{W + \delta_i}, \tag{3.3}$$

where W is the size of the time window (a constant). ICDS chooses the TSW algorithm to smooth the estimation of the arrival rates.

The TSW algorithm has an indispensable division operation over an arbitrary value

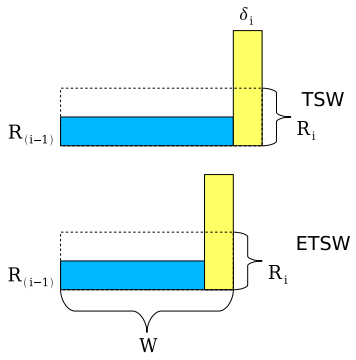


Figure 3.3: Comparison between TSW and ETSW.

$W + \delta_i$. ICDS modifies the TSW algorithm to remove the expensive division operation as follows:

$$R_i = \frac{R_{i-1}(W - \delta_i) + l_i}{W}, \tag{3.4}$$

when $W > \delta_i$; that is, some packets arrive in the last time window). If no packet arrives in the last time window (i.e., $W <= \delta_i$), R_i is set to a small constant R_c . By choosing the constant W with a power of 2, such as 2^w, the division over W can be implemented as a shift operation,

$$R_i = (R_{i-1}((1 \ll w) - \delta_i) + l_i) \gg w. \tag{3.5}$$

where $W = 2^w = 1 \ll w$. This variant of the TSW algorithm is called the *Efficient-TSW* (ETSW).

Clearly, ETSW is only an approximation of TSW. Figure 3.3 illustrates the difference between them. The per-step difference between them can be estimated by comparing the right-hand sides of Equation (3.4) and Equation (3.3) as follows:

$$\frac{R_{i-1}(W-\delta_i)+l_i}{W} - \frac{R_{i-1}W+l_i}{W+\delta_i} = \frac{(l_i - R_{i-1}\delta_i)\delta_i}{W(W+\delta_i)}.$$

This result indicates that the difference between ETSW and TSW is small in each step, if the ratio of the packet inter-arrival time δ_i to the size of the time window W is small.

Furthermore, it is necessary to know whether the accumulated discrepancy between ETSW and TSW breaks the time decaying property in the long run. In the following, a proof similar to the one in [15] confirms that the embedded decaying function in ETSW is independent of the traffic rate, the same as TSW when δ_i is small enough.

Proof: The moving average rate after a time window is evaluated here. Assume all the packets are equal in size. Let l denote the packet size. Assume the rate is constant. Let δ denote the inter-arrival time between the packets. R_i denotes the estimated moving average rate in ETSW when the ith packet arrives. R_0 denotes the initial rate estimation. W denotes the window size. After the first packet arrives, the rate estimation R_1 is updated by

$$R_1 = R_0\left(\frac{W-\delta}{W}\right) + \frac{l}{W}.$$

After the second packet arrives,

$$\begin{aligned} R_2 &= R_1\left(\frac{W-\delta}{W}\right) + \frac{l}{W} \\ &= \left(R_0\left(\frac{W-\delta}{W}\right) + \frac{l}{W}\right)\left(\frac{W-\delta}{W}\right) + \frac{l}{W} \\ &= R_0\left(\frac{W-\delta}{W}\right)^2 + \frac{l}{W}\left(\frac{W-\delta}{W}\right) + \frac{l}{W}. \end{aligned}$$

By induction, after the nth packet arrives,

$$\begin{aligned} R_n &= R_0\left(\frac{W-\delta}{W}\right)^n + \frac{l}{W}\left(\frac{W-\delta}{W}\right)^{n-1} + \frac{l}{W}\left(\frac{W-\delta}{W}\right)^{n-2} + ... + \frac{l}{W} \\ &= R_0\left(\frac{W-\delta}{W}\right)^n + \frac{l}{W}\sum_{i=0}^{n-1}\left(\frac{W-\delta}{W}\right)^i. \end{aligned}$$

The last n items are a geometric sequence ($\Sigma_{i=0}^{n-1} q^i = \frac{1-q^n}{1-q}$). Recall that the value of the moving average R after one TSW window W is to be investigated. Let $W = n\delta$. $\left(\frac{W-\delta}{W}\right)^n$turns to be $\left(1-\frac{1}{n}\right)^n$. Then,

$$\begin{aligned} R_n &= R_0\left(\frac{W-\delta}{W}\right)^n + \frac{l}{W}\left(\frac{1-\left(\frac{W-\delta}{W}\right)^n}{1-\left(\frac{W-\delta}{W}\right)}\right) \\ &= R_0\left(1-\frac{1}{n}\right)^n + \frac{l}{W}\left(\frac{1-\left(1-\frac{1}{n}\right)^n}{1-\left(\frac{W-\delta}{W}\right)}\right) \\ &= R_0\left(1-\frac{1}{n}\right)^n + \frac{l}{\delta}\left(1-\left(1-\frac{1}{n}\right)^n\right). \end{aligned}$$

Clearly, $0 < \left(1-\frac{1}{n}\right)^n < 1$. Let $\alpha = \left(1-\frac{1}{n}\right)^n$, then,

$$R_n = R_0\alpha + \frac{l}{\delta}(1-\alpha). \tag{3.6}$$

The above equation shows that ETSW decays the original rate R_0 by a factor of α after a window length of time W independent of the arrival rate[1]. ■

[1]Equation (3.6) of ETSW is similar to Equation (3.2) of the EWMA algorithm. The difference is that Equation (3.6) is executed every time window and the result is independent of the arrival rate, whereas Equation (3.2) is operated when each packet arrives and the result depends on the arrival rate.

If the ETSW algorithm is applied to relative arrival rates, similar to the simple relative rate estimator (shown in Equation (3.1) on page 26), the variables denoting real time in ETSW (shown in Equation (3.4) on page 29) change to the amount of the received data for all the classes. Conceptually, ICDS does not keep track of real time but uses the total received data as time (in terms of bytes and called *byte time*). W is the size of the byte-time window. δ_i is the received data of all the classes (i.e., the elapsed byte time) between the receptions of the ith packet and the $(i-1)$th packet.

Because ETSW is operating on byte time instead of real time, the decaying of ETSW is dependent on the total arrival rate (rather than the individual arrival rate of a class). If the total arrival rate is high, ETSW forgets the rate history faster, and vice versa. However, this is different from weight-based estimators because the ETSW estimators of all the classes forget their rate histories at the same speed. Therefore, the ETSW estimators estimate the average relative arrival rates in the time windows with the same size even when the traffic rates are different.

3.1.3 Frequency of Rate Estimation

ICDS can adjust the service rates of the classes in proportion to their arrival rates at per-packet processing. One drawback with such an approach is that a class with a higher arrival rate adjusts its service rate more frequently than a class with a lower arrival rate does. In such scenario, the low-speed class can miss the chance to increase its rate (a simulation result illustrating the consequence is shown in Section 4.4 on page 57). A fair approach is to estimate arrival rates and adjust service rates with the same time interval (called *update interval*) for all the classes.

To avoid timer support, ICDS estimates relative rates and adjusts service rates every few packets. The elapsed time between the two packet arrivals with service-rate adjustments is

if $t - T_{k-1} < C$ **then**
  $L_k = L_k + l_i$
else
  $T_k = t$
  $\Delta_k = T_k - T_{k-1}$
  if $\Delta_k \geq W$ **then**
    $R_k = R_c$
  else
    $R_k = \frac{R_{k-1}(W-\Delta_k)+L_k}{W}$
  end if
end if

R_k	the kth estimated relative rate
t	current byte time
T_k	the byte time when the service rate is adjusted the kth time.
L_k	the amount of data received from T_{k-1} to T_k
Δ_k	the byte-time interval between T_{k-1} and T_k
C	the update interval, a constant

Figure 3.4: Arrival-rate estimation in a time interval.

approximately equal to the configured update interval C by the following simple approach. ICDS maintains the elapsed time Δ since the last service-rate adjustment. When a packet arrives, ICDS compares Δ with C. If Δ is larger than C, ICDS updates the estimation of the arrival rate and adjusts the service rate of this class. Otherwise ICDS does nothing.

Figure 3.4 shows the revised per-packet processing algorithm in each traffic class. The rate estimator (Equation 3.4 on page 29) is modified to the version with a update interval: the packet inter-arrival time δ_i is replaced by the elapsed time between the adjacent rate adjustments Δ_k; the size l_i of the ith packet is replaced by L_k, the amount of data received in Δ_k.

The update interval C should not be larger than the size of the time window W, because ETSW requires $C < \Delta_k < W$ most of the time (otherwise, the relative arrival rate is set to a constant R_c as shown Figure 3.4). C should not be too small either. Ideally, each class adjusts its service rate with the update interval C. However, C is approximated by Δ_k in this algorithm. The error between C and Δ_k increases when C decreases. For an extreme example, if C is set to be less than the packet inter-arrival time, this algorithm degrades to the original version which adjusts service rates when each packet arrives.

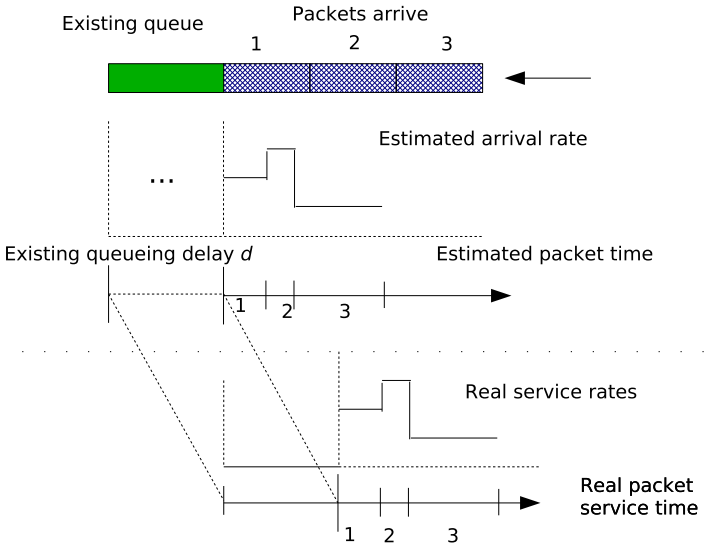


Figure 3.5: Control delay and packet time.

3.2 Service-Rate Adjustment Delay

When a packet arrives, ICDS estimates the queueing delay of this packet and places it at the end of the queue if this packet can be transmitted within the delay target. Otherwise, ICDS drops this packet. If the arrival rate changes, ICDS adjusts the service rate. Ideally, ICDS adjusts the service rate immediately after the arrival rate change. However, if the queue is not empty, ICDS has to delay the service-rate adjustment from the time when the arrival rate is estimated (i.e., a packet arrives) to the time when this packet is served (i.e. when the packet goes to the head of the queue). The reason is that a service rate change (in particular, a service rate decrease) when a new packet arrives would break the previous service-rate allocation for the accepted packets in the queue, thereby violate the delay guarantee of these packets. Figure 3.5 illustrates that ICDS allocates the service rate of each packet when it arrives at the head of the queue.

Intuitively, ICDS can estimate the queueing delay of a class by dividing the queue

length by the service rate. However, the estimated arrival rate, and therefore, the service rate for calculating the service time can change within the time interval of the queueing delay. ICDS maintains the service time of a packet, called *packet time*, and calculates the queueing delay as the sum of the packet times of all the packets in the queue. Figure 3.5 shows an example where the estimated arrival rates of packets differ. Obviously, it is not necessary to maintain the service time in a granularity finer than a packet time because the estimated arrival rates only change when ICDS processes the packets.

ICDS can be viewed as a control system. The arrival rates are the input; the service-rate adjustments are the feedback. For each class, the delay of the feedback introduces a control delay with a value of the current queueing delay. This delay is inevitable, but the simulation results in Chapter 4 prove that the overall behaviour is acceptable.

3.3 Rate Budget for Strict Delay Target

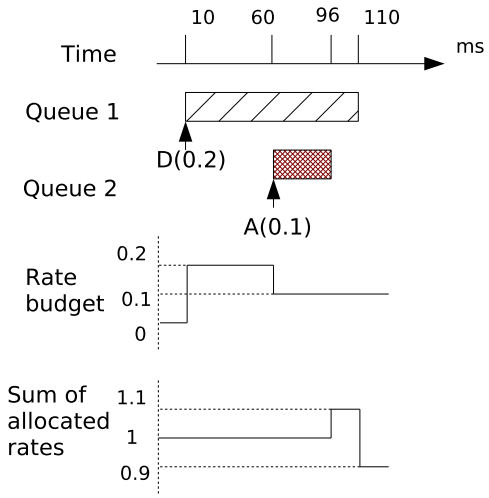
In ICDS, the delay experienced by packets can violate the delay target without a service-rate allocation check. Why? The sum of the estimated relative arrival rates of all the classes may exceed 1.0 because different classes measure the relative arrival rates at different time intervals. Therefore, the allocated service rates could temporarily exceed the total bandwidth offered by the link. The fairness property of the packet GPS scheduler ensures that all the classes receive a service rate less than what is expected if the link bandwidth cannot accommodate the total arrival rate. Therefore, the calculated service time of a packet (by dividing the packet length by the estimated service rate) would be less than the real service time. The estimated queueing delay (the sum of the service times of the packets in the queue) would be less than the real queueing delay. This would result in that ICDS admits this packet which should be dropped, because although the estimated

queueing delay does not exceed the delay target, the real queueing delay does.

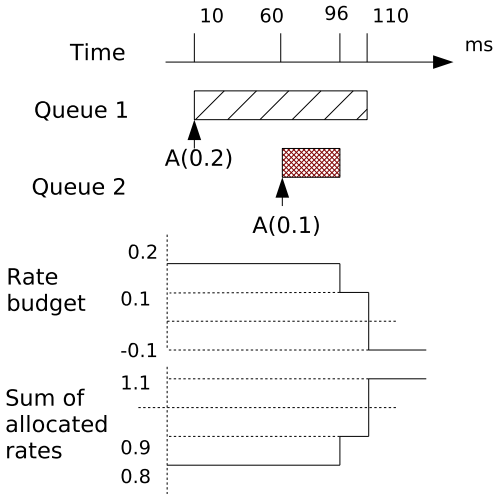
From the previous discussion, it is clear that the sum of the service rates to be allocated should be less than the link bandwidth, if strict delay targets are desired. ICDS guarantees this by maintaining a variable to record the value of the available link bandwidth, called the *rate budget*. If the service rate (i.e., the estimated relative arrival rate) of a class decreases, ICDS releases the rate difference to the rate budget. If a class requests more service rate and sufficient rate budget exists, ICDS allocates the rate difference from the rate budget. Otherwise, ICDS does not increase the service rate allocation.

Because the control delay exists between the arrival-rate estimation and the service-rate adjustment, it is not obvious when to allocate the rate difference from the rate budget (called *allocation*) and when to release the rate difference to the rate budget (called *release*). Three strategies are now considered to guarantee that the sum of the allocated service rates is less than the link bandwidth at any time.

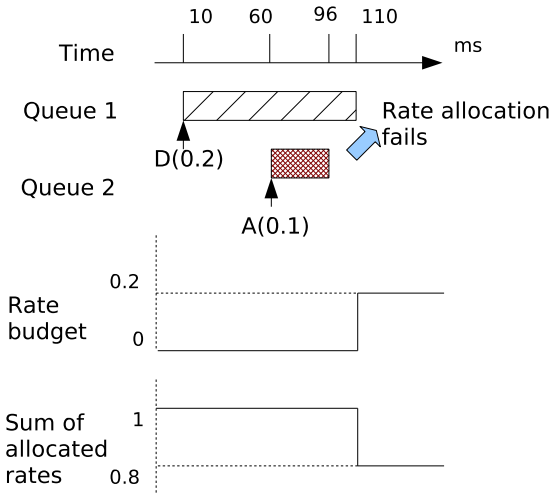
1. **Allocation and release when ICDS estimates arrival rates.** If the arrival rate of a class decreases, the rate difference is released to the rate budget. However, the allocated service rates does not change until all the packets in the queue are served. In the time period between the release and the realization of the rate decrease, ICDS may falsely admit a new allocation as shown in Figure 3.6(a). The queueing delays of queue 1 and queue 2 are 100ms and 36ms respectively. ICDS releases 0.2 from class 1 at 10ms and allocates 0.1 to class 2 at 60ms. The sum of the allocated rates exceeds the link bandwidth after ICDS serves the end of queue 2. The reason is that the sum of allocated service rates does not decrease at 10ms, whereas the rate budget increases at that time. ICDS falsely admits the allocation of Queue 2 at 60ms because the rate budget indicates that the available bandwidth is sufficient. This strategy does not guarantee that the sum of the allocated rates is less than or equal to the



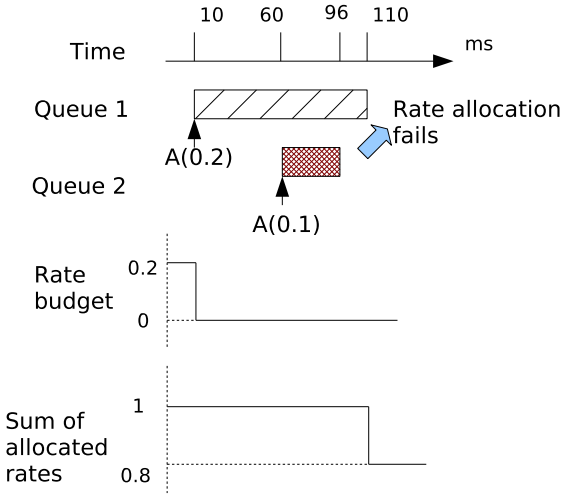
(a) Allocation and release when rates are estimated. The initial rate budget is 0 and the initial sum of allocated rates is 1.



(b) Allocation and release when service rates are allocated. The initial rate budget is 0.2 and the initial sum of allocated rates is 0.8.



(c) Allocation at rate estimation; release at rate adjustment.



(d) Allocation at rate estimation; release at rate adjustment.

Figure 3.6: Strategies of *allocations* and *releases* of the rate budget. A(r) represents that ICDS allocates r from the rate budget. D(r) represents that ICDS releases r to the rate budget. (c) and (d) are the operations of the third strategy on the examples of (a) and (b) respectively.

bandwidth at all time.

2. **Allocation and release when ICDS adjusts service rates.** The availability of the rate budget is used to judge whether to rate increase should be accepted. If allocation and release are both delayed to the time when ICDS adjusts service rates, the rate budget does not contain the information of the most recent allocations. Figure 3.6(b) gives an example. ICDS accepts the rate allocation of both class 1 and 2. The sum of the allocated rates increases to 1.1 at 110ms. The reason is that ICDS falsely accepts the allocation of class 2 because ICDS does not detect the allocation of class 1 is already made at 60ms. This strategy does not guarantee the sum of the allocated rates is less than or equal to the bandwidth at all time either.

3. **Allocation at arrival-rate estimation and release at service-rate adjustment.** This strategy addresses the problems of the first two strategies. Allocation at the time of arrival-rate estimation memorizes any potential increase of the sum of the allocated rates. Release at the time of the service-rate adjustment guarantees that the decrease of the sum of the allocated rates is realized. Figure 3.6(c) and (d) illustrate the operations of this strategy on the examples of Figure 3.6(a) and (b) respectively. The sum of the allocated rates never exceeds the link bandwidth.

ICDS uses the third strategy. It may delay some rate allocations, although the available bandwidth is sufficient because of the control delay. The allocation is deducted from the rate budget at the time of rate estimation whereas the actual service-rate allocation is made when the packet becomes the head of the queue. Figure 3.7 provides such an example. Although the available bandwidth is sufficient, the allocation of queue 3 fails. The allocation of queue 3 is delayed to a later rate-adjustment opportunity after the release of queue 1 is realized. The first several packets of class 3 are dropped unnecessarily. However,

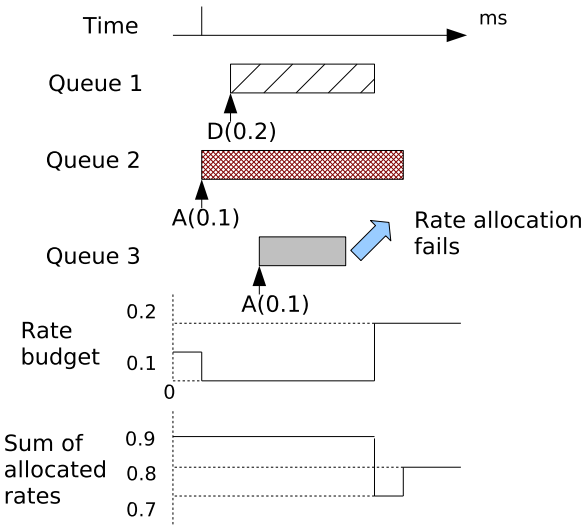


Figure 3.7: *Allocation* fails, but the link bandwidth is sufficient in the third strategy.

such packet drops are few because the sum of the estimated relative arrival rates is close to 1.0.

As discussed above, the benefit of the rate-budget mechanism have a price. With the rate-budget mechanism, ICDS sacrifices some utilization for strict delay guarantee. A better variant of ICDS is an hybrid system that provides both strict delay classes for delay-sensitive applications and loose delay classes (the long-term average delay is within the delay target whereas the instantaneous delay can exceed the delay target) for TCP applications without loosing utilization. This is left for future work.

3.4 Division Using Reciprocal

ICDS could calculate the packet time by dividing the packet length by the estimated relative arrival rate. Because division operations are expensive, alternatively, the packet time can be computed by multiplying the packet length by the reciprocal of the estimated

relative arrival rate. ETSW estimates the reciprocal of the arrival rate R' similarly to estimate the arrival rate in Equation (3.4) on page 29 with l_i and δ_i switched as follows:

$$R'_i = \frac{R'_{i-1}(W - l_i) + \delta_i}{W}. \tag{3.7}$$

If no packets arrive in the last window W, R'_i is set at a constant $1/R_c$ where R_c is the constant used in the arrival-rate estimator under the same condition (described in Equation (3.4) on page 29). If the arrival rate is constant, the reciprocal R'_i is always the same as $1/R_i$, because the smoothing algorithm does not have an impact on a constant. However, if the arrival rate varies, they are not identical.

Another usage of R' is to adjust the service rates. A packet GPS scheduler estimates the service time of a packet by dividing its length by the service rate [12, 37]. This dividing operation should also be replaced with a multiplication operation for efficiency in the real-world implementation. The packet time of a packet is calculated by multiplying its packet length by the reciprocal of the service rate in the packet GPS scheduler of ICDS.

Because ICDS uses R' to adjust service rates, the total allocated service rate is $\sum 1/R'_i$. However, to avoid any division operations, ICDS uses R to calculate the rate budget $(1 - \sum R_i)$ such that the total *estimated* allocated rates is $\sum R_i$. If the estimated total service rate $\sum R_i$ is less than the allocated total service rate $\sum 1/R'_i$, the allocated total service rate $\sum 1/R'_i$ is larger than the link bandwidth (the rate budget mechanism essentially makes $\sum R_i$ equal to the link bandwidth). Consequently, as shown in the discussion of the design goal of the rate budget (Section 3.3), some packets will experience a queueing delay longer than the delay target. Similarly, if the estimated total service rate $\sum R_i$ is larger than the allocated total service rate $\sum 1/R'_i$, the queueing delay will be shorter than expected.

The maximal difference between $\sum 1/R'_i$ and $\sum R_i$ is shown on the case of two classes

in the following discussion. R_1 and R_2 denote the original relative rates ($R_1 + R_2 = 1$ holds) of class 1 and class 2 respectively. L_1 and L_2 denote the amount of the received data in class 1 and class 2 in the next update interval Δ respectively ($L_1 + L_2 = \Delta$ holds). At the next rate estimation, the sum of the estimated relative arrival rates $S(R_1, L_1)$ is defined as a function of R_1 and L_1:

$$S(R_1, L_1) = \frac{R_1(W - \Delta) + L_1}{W} + \frac{R_2(W - \Delta) + L_2}{W}. \tag{3.8}$$

Similarly, the total allocated service rate $RS(R_1, L_1)$ is defined as a function of R_1 and L_1:

$$RS(R_1,L_1) = \frac{1}{\frac{1/R_1(W-L_1)+\Delta}{W}} + \frac{1}{\frac{1/R_2(W-L_2)+\Delta}{W}}, \tag{3.9}$$

where R_1 ranges between 0 and 1 and L_1 ranges between 0 and Δ. $S(R_1, L_1)$ is always 1 by the following simple algebra manipulation:

$$\begin{aligned} S(R_1, L_1) &= \frac{R_1(W - \Delta) + L_1}{W} + \frac{R_2(W - \Delta) + L_2}{W} \\ &= \frac{(R_1 + R_2)(W - \Delta) + (L_1 + L_2)}{W} \\ &= \frac{W - \Delta + \Delta}{W} = 1. \end{aligned}$$

Therefore, to find the maximal difference between $RS(R_1, L_1)$ and $S(R_1, L_1)$ is equivalent to find the maximum or the minimum of $RS(R_1, L_1)$. A necessary condition where the maximum or the minimum of $RS(R_1, L_1)$ locates is that all the partial derivatives of $RS(R_1, L_1)$ vanish (the stationary point) as follows:

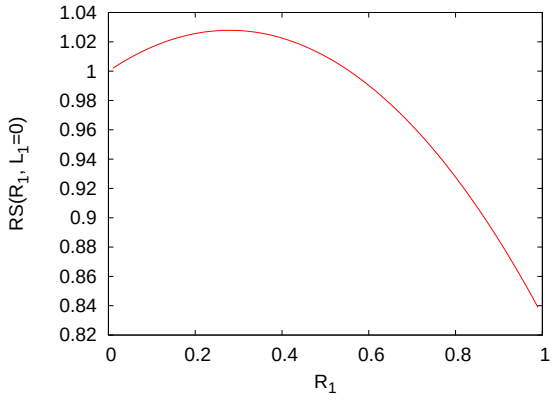


Figure 3.8: The value of $RS(R_1, L_1)$ when $L_1 = 0$. ($W = 20$, $\Delta = 4$).

$$\begin{aligned} \frac{\partial RS(R_1, L_1)}{\partial R_1} &= 0, \\ \frac{\partial RS(R_1, L_1)}{\partial L_1} &= 0. \end{aligned}$$

The analytical solution of the system of these two equations is $R_1 = \frac{1}{2}$ and $L_1 = \frac{\Delta}{2}$ as solved by Maple [2]. The value of $RS(R_1, L_1)$ at $(\frac{1}{2}, \frac{\Delta}{2})$ is 1. Clearly, it is neither the maximum nor the minimum of $RS(R_1, L_1)$. Therefore, the maximum and the minimum fall on the boundaries (the lines: $L_1 = 0$, $L_1 = \Delta$, $R_1 = 0$ and $R_1 = 1$). After these boundaries are examined, it turns out that the maximum falls on the line $L_1 = 0$ or $L_1 = \Delta$. Figure 3.8 shows the value of $RS(R_1, L_1)$ on the line $L_1 = 0$.

If the ratio of the TSW window size W and to update interval Δ is σ (i.e., $\sigma = \frac{W}{\Delta}$), then the maximum of $RS(R_1, L_1)$ is the following (the value where $\frac{\mathrm{d}RS(R_1,1)}{\mathrm{d}R_1} = 0$ on the

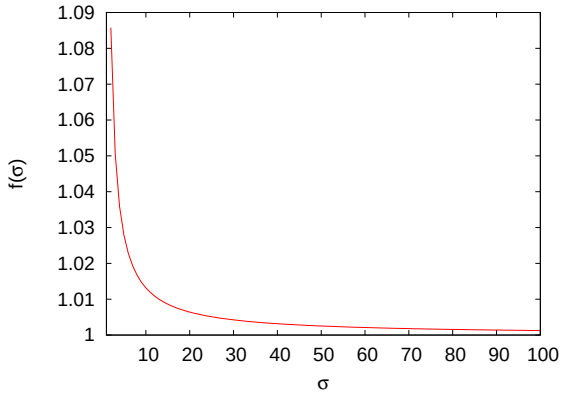


Figure 3.9: The maximum of $RS(R_1, L_1)$ as a function of σ (the ratio of W to Δ).

line $L_1 = 0$) as solved by Maple:

$$f(\sigma) = \frac{\sigma}{\frac{1}{2\sigma-1-2\sqrt{\sigma^2-\sigma}}+1} + \frac{\sigma}{\frac{\sigma-1}{1-\left(2\sigma-1-2\sqrt{\sigma^2-\sigma}\right)\sigma}+1}, \tag{3.10}$$

when $R_1 = \left(2\sigma - 1 - 2\sqrt{\sigma^2-\sigma}\right)\sigma$ and $L_1 = 0$.

Figure 3.9 illustrates that $f(\sigma)$ decreases and approaches 1 infinitely close (i.e., the error of $(RS(R_1, L_1) - S(R_1, L_1))$ decreases) when σ grows. For a numerical example, if $\sigma = 5$, the value of $f(\sigma)$ is 1.028; $R_1 = 0.279$ and $L_1 = 0$; that is, if the rate changes from 0.279 to 0, the sum of the estimated relative arrival rates may be 2.8% less than the sum of the allocated service rates. Some packets can experience a queueing delay longer than the estimated queueing delay by 2.8%. To guarantee a strict delay bound, a simple solution is to reduce the total available rate budget by $2.8/102.8 = 2.7\%$; that is, the total rate budget is 97.3%.

Similarly, the minimum of $RS(R_1, L_1)$ falls on $(0, \Delta)$ or $(1, 0)$ (when the rate of one

class changes from 0 to 1 or from 1 to 0). The limit of the minimum as a function of σ is

$$g(\sigma) = \frac{\sigma}{\sigma + 1}. \tag{3.11}$$

Clearly, given $\sigma > 1$, $g(\sigma) < 1$; $g(\sigma)$ grows and approaches 1 when $\sigma \to \infty$. $g(\sigma)$ is 0.83 if $\sigma = 5$. The error in most cases is smaller because the minimum of $RS(R_1, L_1)$ can be reached only when the relative rate of a traffic class changes dramatically from 0 to 1 or vice versa. Some packets may be dropped unnecessarily because the real queueing delay is less than the estimated queueing delay.

3.5 Convergence

It is important to investigate the behaviour of ICDS with TCP traffic because TCP traffic dominates the Internet. This section discusses the design mechanisms to improve the convergence of TCP traffic on ICDS. A new TCP flow uses congestion control algorithms (i.e., *slow start* and *congestion avoidance* [5]) to detect its fair share of the available bottleneck link bandwidth and determine its sending rate. Furthermore, all the other TCP flows need to re-detect their fair shares and adjust their sending rates when a TCP flow joins the competition for the link bandwidth. Consequently, there exists a time interval (called *convergence time*) between the time when a TCP flow starts (or ends) its transmission and the time when the sending rates of all the TCP flows are stable again in the ICDS system.

TCP traffic converges fast for FIFO DropTail queueing (Section 4.2 shows the simulation result) which is widely deployed in the Internet. One of the design goal of ICDS is to achieve a convergence time comparable to DropTail queueing. Two mechanisms help the convergence of TCP traffic in ICDS. They are described in the flowing.

3.5.1 Minimal Rate Mechanism

The initial sending rate of a TCP traffic class[2] is zero at the moment when it starts to transmit data in ICDS. Consequently the initial service rate is zero for this class because ICDS allocates the link bandwidth in proportion to the arrival rates of all the classes. A zero service rate prevents the first packet (or several packets) of TCP to be transmitted. Recall that TCP is a feedback congestion control protocol. If no ACK is received for the first packet, TCP is unable to send subsequent packets. Therefore, a new TCP class would never succeed to transmit any data, if ICDS strictly obeys the rate allocation rule. The convergence time would be infinite.

Configuring a minimal rate c for each class addresses this problem. The value of c must be large enough to transmit the packets in the initial congestion window of TCP within the delay target.

The number of TCP flows that start transmission can have a large range of values. However, the minimal rate c is a constant. A relative small c is acceptable in practice because TCP flows in one class are not likely to begin transmission at the same time. However, the packets in the initial congestion windows have a higher chance of being dropped, if the number of flows that start transmission within a short time interval becomes significantly large, which results in a slow convergence.

3.5.2 Add-Rate Option

ICDS reduces convergence time by another simple mechanism called the *add-rate option*. The following requirements are necessary to boost convergence:

1. The mechanism should be biased to allocate more service rates to a class that needs

[2] A TCP traffic class contains single or multiple TCP flows. Its sending rate is the aggregate sending rates of all the TCP flows in it.

to increase rate to reach the stable state, but allocate less service rates to a class that needs to decrease its rate to reach the stable state.

2. The mechanism should not be biased when the system is stable.

The add-rate option reduces the convergence time by allocating the service rate s_i of class i in proportion to the sum of the arrival rate r_i and a constant e as follows:

$$\frac{s_i}{s_j} = \frac{r_i + e}{r_j + e}, \tag{3.12}$$

where s_j and r_j are the service rate and the relative arrival rate of class j respectively.

The following discussion describes an example to illustrate the effectiveness of the add-rate option in a ICDS system with two TCP classes. Initially, the traffic from TCP class 1 occupies all the link bandwidth. Now the flows of TCP class 2 enter the system and start transmission. The system begins to shift to a stable state in which each TCP class receives one half of the link bandwidth, if the delay targets of these two TCP classes are the same. In the transition stage, the arrival rate of TCP class 2 is less than the arrival rate of TCP class 1 (i.e., $r_1 > r_2$). TCP class 2 needs to increase its rate, whereas TCP class 1 should decrease its rate.

The service share $\frac{r_2+e}{r_1+r_2+2e}$ of TCP class 2 with the add-rate option is larger than its share $\frac{r_2}{r_1+r_2}$ without the add-rate option as follows:

$$\begin{aligned} \frac{r_2 + e}{r_1 + r_2 + 2e} - \frac{r_2}{r_1 + r_2} &= \frac{(r_1 + r_2)(r_2 + e) - r_2(r_1 + r_2 + 2e)}{(r_1 + r_2 + 2e)(r_1 + r_2)} \\ &= \frac{(r_1 - r_2)e}{(r_1 + r_2 + 2e)(r_1 + r_2)} > 0. \end{aligned} \tag{3.13}$$

Similarly, the service share of TCP class 1 is less than its share without the add-rate option. The add-rate option also satisfies the second requirement for a fast convergence. If $r_1 = r_2$, both TCP classes receive half of the link bandwidth and no bias exists. Similar examples can be provided to show that the add-rate option also works for multiple TCP classes.

In addition, Equation (3.13) shows that, if e grows, the bias is larger. Therefore, the convergence time is reduced more. However, as e becomes larger, ICDS deviates more from its original design: allocating bandwidth proportional to the arrival rates. Consequently, e should be a trade-off between fast convergence and the original design goal.

ICDS merges the add-rate option into the relative-rate estimator. The rate estimator in Figure 3.4 on page 33 is changed to

$$R_k = \frac{R_{k-1}\left(W - (\Delta_k + Ne)\right) + (L_k + e)}{W}, \tag{3.14}$$

where N is the number of active classes. The estimator of the reciprocals of relative arrival rates is modified similarly.

Figures 3.10, 3.11, and 3.12 provide the pseudocode of the rate estimation, the enqueue operation, and the deque operation of ICDS.

```
if (now-last[i]) < updateInterval then
  data[i] ← data[i] + size
else
  if (now-last) > (1 ≪ w) then
    rate[i] ← minRate
    recipRate[i] ← recipMinRate {recipMinRate = 1/minRate}
  else
    rate[i] ← (rate[i] * ( (1 ≪w) − (now - last[i] + n ∗ e) ) + (data[i] + e) ) ≫ w
    recipRate[i] ← (recipRate[i] * ((1 ≪w) − (data[i] + e) ) + (now − last[i] + n ∗ e)
    )) ≫ w
    {if rate is less than the minimal rate, set to it.}
    if rate[i] < minRate then
      rate[i] ← minRate
      recipRate[i] ← recipMinRate
    end if
    last[i] ← now
    data[i] ← 0
  end if
end if
```

Figure 3.10: Pseudocode of the rate estimation: **procedure rateEstimate(i, now, size)**, where "i" is the classID, "now" is the current byte time, and "size" is the size of the packet received. The TSW window size is 2^w (i.e., $(1 \ll w)$), e is the add-rate constant, and n is the number of active classes.

```
now ← now + packet.size
i ← packet.classID
oldRate ← rate[i]
recipOldRate ← recipRate[i]

{Update rate[i] and recipRate[i].}
rateEstimate(i, now, packet.size)

rateDelta ← rate[i] − oldRate

{If rateBudget is not sufficient, revert to the old rate.}
if  rateDelta> 0 and rateDelta > rateBudget  then
   rate[i] ← oldRate
   recipRate[i] ←recipOldRate
   rateDelta ← 0
end if

{Calculates the packet service time by multiplication by the reciprocal}
packetTime ← packet.size * recipRate[i]

{Drop this packet if it cannot be served within the delay target.}
if  (que[i].delay + packetTime) > delayTarget[i]  then
   drop(packet)
else
   que[i].delay ← que[i].delay + packetTime

   {Carry information with packet.}
   packet.time ← packetTime
   packet.rateDelta ← rateDelta
   packet.rate ← rate[i]
   que[i].enque(packet)

   { "Allocation" the rate difference from the rate budget when estimate the arrival rates.}
   if rateDelta ¿ 0 then
      rateBudget ← rateBudget − rateDelta
   end if
end if
```

Figure 3.11: Pseudocode of the enque operation: **procedure enque(packet)**.

```
i ← nextClassID() {Get the id of next class to serve.}
packet ← que[i].deque()
que[i].delay ← que[i].delay − packet.time

{"Release" the rate difference to the rate budget when adjust service rates.}
if packet.rateDelta < 0 then
   rateBudget ← rateBudget − packet.rateDelta
end if

{Delay the adjustment of the service rate until this packet becomes the head of the queue.}
GPSClass[i].setRate(packet.rate)
send(packet)
```

Figure 3.12: Pseudocode of the deque operation: **procedure deque()**.

Chapter 4

Evaluation

This chapter presents the analysis and experiments to investigate the behaviour of ICDS. These analysis and simulations try to answer the following questions. Can ICDS provide separation between smooth traffic and bursty traffic? Does bursty traffic need more buffering, therefore a larger delay target, than smooth traffic? Does an optimal delay target exist for a special type of traffic? These simulations also support the arguments in Chapter 3, such as the effect of the add-rate option and the frequency of service-rate adjustment.

This chapter is organized as follows. Section 4.1 introduces the simulation configuration. Section 4.2 shows the impact of the min-rate mechanism and the add-rate option on the convergence time of TCP traffic in ICDS. Section 4.3 illustrates the proper TSW window size. Section 4.4 presents the experiments to show that it is important to update the rate estimations and adjust the service rates for all the classes at the same frequency in ICDS. Section 4.5 presents the analysis and simulation to study the behaviour of ICDS with different types of traffic.

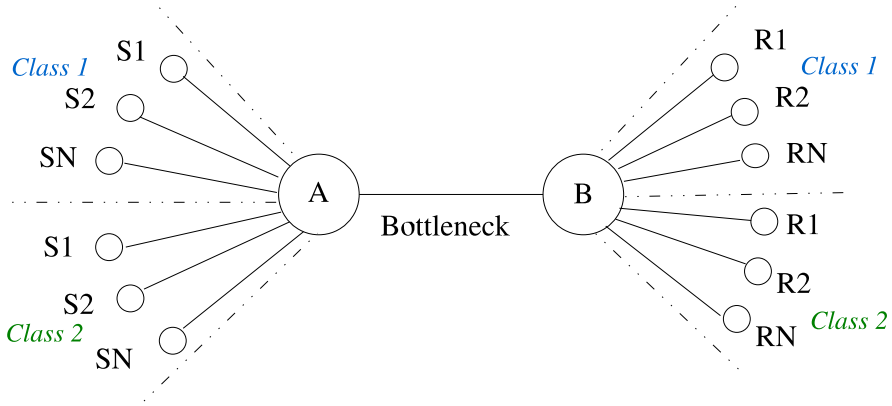


Figure 4.1: The standard configuration of the simulations. Two classes that contain multiple flows compete for a bottleneck link.

4.1 Simulation Configuration

The simulations in this thesis use the Network Simulator (ns-2) [3]. All experiments employ a standard configuration to study how ICDS differentiates two traffic classes as shown in Figure 4.1. The topology of this configuration is the dumbbell topology. Two classes that contain multiple flows compete for a bottleneck link.

Table 4.1 summarizes the frequent used simulation parameters. The exceptions of them are pointed out in each simulation. All the simulation results shown in this chapter use TCP Newreno [27]. Several experiments are conducted with other TCP variants. It is found the throughput of TCP Sack [33] is smoother than TCP Newreno in the simple two-class case, whereas the throughput of TCP Reno [30] and TCP Tahoe [29] is burstier. However, no significant differences are currently found on the average throughput and loss rates between TCP Sack, TCP Reno and TCP Newreno in the context of following studies. Some differences on throughput and loss rates exists between TCP Tahoe and TCP Newreno. But with the rapid deployment of TCP Newreno [18] and TCP Sack [19],

Packet size	1000 bytes
Propagation delay from A to B	10ms
Bandwidth from A to B	20Mbps
Propagation delay from S_i to A, B to R_i	10ms
Bandwidth from S_i to A, B to R_i	20Mbps
Number of flows for each traffic class	10
Simulation Length	200 seconds
TSW window size	the maximal delay target
Update interval	1/5 of the TSW window size
Minimal Rate	3MSS/(the minimal delay target)

Table 4.1: Summary of simulation parameters. The size of a Maximum Segment Size (MSS) is set to 1500 bytes in the simulations.

the behaviour of TCP Tahoe with ICDS is not relevant.

TCP Flows are started randomly to avoid deterministic phenomena that never happen in the real world. Random processing time is added to the sender to avoid the traffic phase effect of TCP as suggested in [21].

In this chapter, some simulation results are shown with only one replication because they are relatively deterministic and the confidential interval of multiple replications is very small. For example, the simulation results with TCP traffic and CBR UDP traffic. Other simulations are conducted with 10 replications with different random seeds; the average value and the 95% confidential interval are shown because there exists at least one random component with a strong impact on the simulation result. For example, the simulation results of self-similar traffic and short web-like TCP traffic.

4.2 Convergence Time of TCP Classes

As defined in Section 3.5, the convergence time is the time interval between the time when the flows in a TCP class start or stop transmission to the time when the system is stable again. This section presents the experiments to study the convergence behaviour of TCP traffic in ICDS. The simulation result of DropTail Queueing in the same environment is shown as a benchmark.

In these simulations, the standard simulation configuration of Figure 4.1 is employed. Two TCP classes with 10 TCP flows share a bottleneck link. The TCP flows in class 1 start randomly between 0 and 5 seconds; the TCP flows in class 2 begin randomly between 10 and 15 seconds. The rule-of-thumb buffer size for a router is the delay-bandwidth product (i.e., $2T_p * C$, where T_p is the one way propagation delay and C is the bottleneck link capacity) [42]. Therefore, the buffer size for DropTail queueing is set at 150 packets (60ms*20Mbps/1000bytes). The delay targets of the two TCP classes in ICDS are both set to the round trip propagation delay (60ms) such that the total buffer size for them is the delay-bandwidth product if they share the bottleneck link bandwidth equally.

Figure 4.2 shows the throughput of two classes between 0 and 100 secs. Each point in the figures represents the average throughput in 0.5 seconds. The convergence time of a simulation is the time interval from roughly 10 seconds (i.e., when the flows in class 2 start transmitting data) to the time that the throughput of the two classes are relative stable.

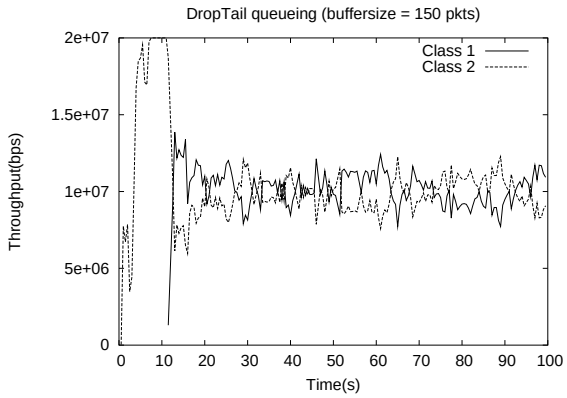
Figure 4.2(a) shows the behaviour of DropTail queueing. The two TCP classes converges quickly with a convergence time of less than 5 seconds. Without the minimal mechanism, the flows in class 1 can not start transmission. The simulation result is not shown here. Figure 4.2(b) demonstrates the behaviour of ICDS with only the minimal-rate mechanism. The throughput of the two traffic classes converges. However, the convergence time (approximately 40 seconds) is much larger than the DropTail Queueing. Figure 4.2(c) displays

the behaviour of ICDS with both the min-rate mechanism and the add-rate option. The add-rate constant is set at 200kbps, 1MSS/(the minimal delay target). The convergence time is reduced to about 10 seconds which is comparable to the convergence time of the DropTail Queueing.

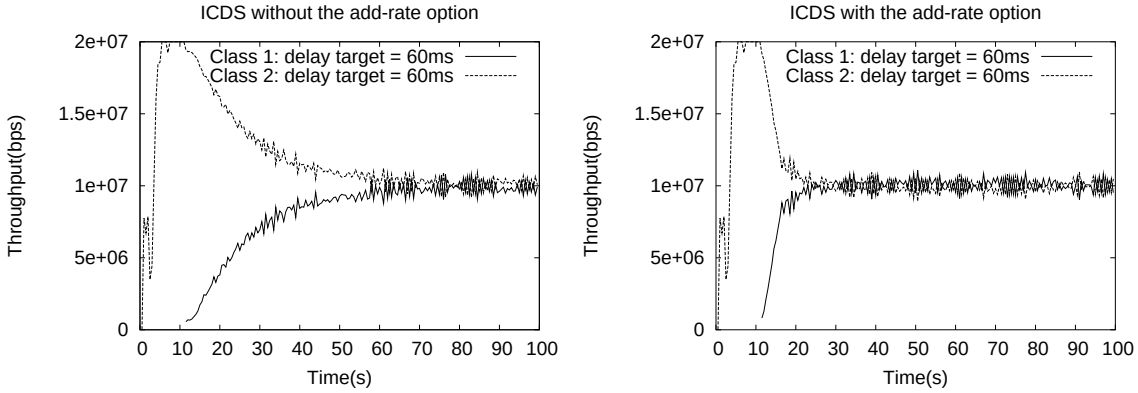
These experiments show that the add-rate option does reduce the convergence time of the TCP traffic in ICDS to a value that is comparable to the one on DropTail Queueing. If there is TCP traffic, the minimal-rate mechanism is indispensable. If fast convergence is preferred, it is better to enable the add-rate option.

4.3 Size of the TSW Window

The size of the TSW window is a trade-off between a fast response and the stability of the system. If the window size is too large, ICDS responds slowly to the varying arrival rates; if the window size is too small, the system is not stable. The time length of TCP's feedback loop is the round trip time (RTT). Therefore, if TCP traffic exists, the size of the TSW window in ICDS must be comparable to the average round trip time of all the TCP classes. Figure 4.3(a) shows that the convergence time is large if the size of TSW window is 5 times the average RTT; Figure 4.3(b) shows that the two TCP classes do not converge if the TSW window size is 1/10 of the average RTT. Figure 4.3(c) show the simulation result when the RTT of different flows are different. In this experiment, the backbone link delay is 1ms. The access link delays are generated from a uniform random distribution between the time interval [1ms, 20ms]. Therefore, the round trip propagation delays of different flows are randomly distributed between 4ms and 82ms. The size of the TSW window is 1/2 of the average RTT. The simulation result shows that the throughput of TCP flows converges. These three experiments verify the analysis that the TSW window must be



(a)



(b) (c)

Figure 4.2: Convergence time of the two TCP classes.

comparable to the average RTT of all TCP classes.

4.4 Same Updating Frequency

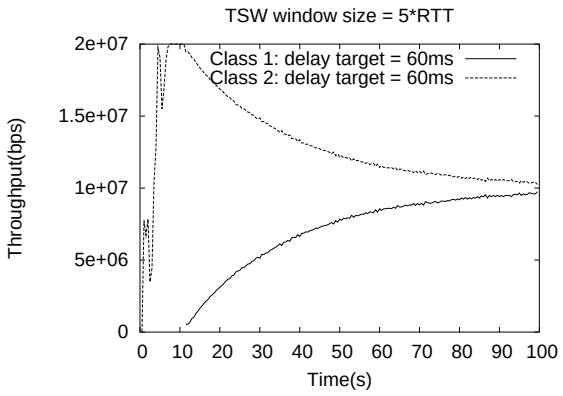
The TCP classes update their arrival rates and adjust their service rates at the same frequency independent of the arrival rates in the previous experiments. These experiments have demonstrated that the rates of the TCP classes converges. Figure 4.4 shows the results when the TCP classes adjust their service rates in per-packet processing with both the minimal-rate mechanism and the add-rate option. The result is that the TCP traffic does not converge to the ideal state where each class shares half of the link bandwidth within 100 seconds. This occurs because high speed traffic has more chance to adjust its service rate than low speed traffic. Consequently, low speed traffic cannot capture all the chances to increase its service rate, when the bandwidth is available. ICDS must update rate estimation and adjust service rates for all the classes in a same frequency.

4.5 Behaviour of ICDS

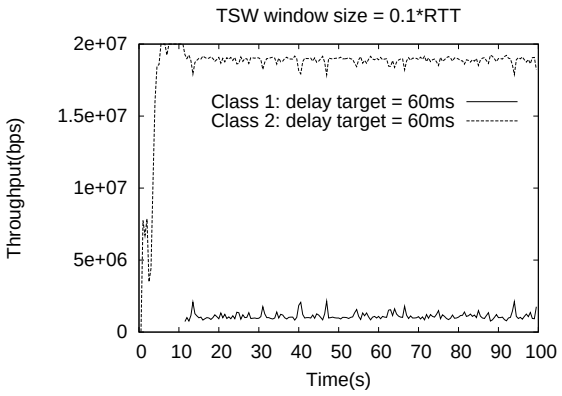
This section studies the behaviour of ICDS with different types of traffic.

4.5.1 Long TCP Flows

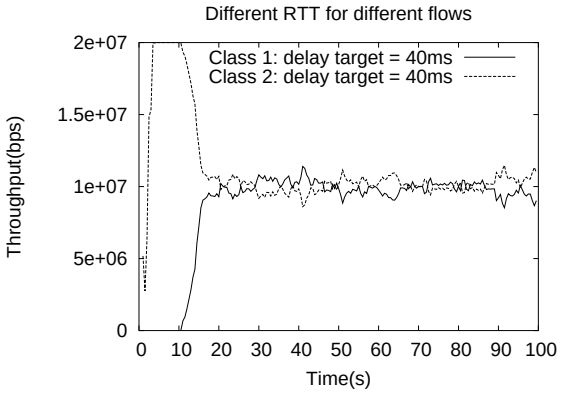
TCP's throughput equation, Equation (2.3) on page 12, indicates that the throughput of TCP is inversely proportional to its RTT and the square root of its loss rate. The number of TCP flows determines how aggressive the aggregate TCP traffic is [34]. However, it is unclear how the "single-class" TCP behaviour with DropTail queueing is related to the TCP behaviour with ICDS where multiple TCP classes exist. The case of two TCP classes



(a) System converges slowly.



(b) System does not converge to the ideal stable state.



(c) The delays of the access links are randomly generated from a uniform distribution between 1ms and 20ms. TSW window size = 0.5* (average RTT).

Figure 4.3: TSW window size.

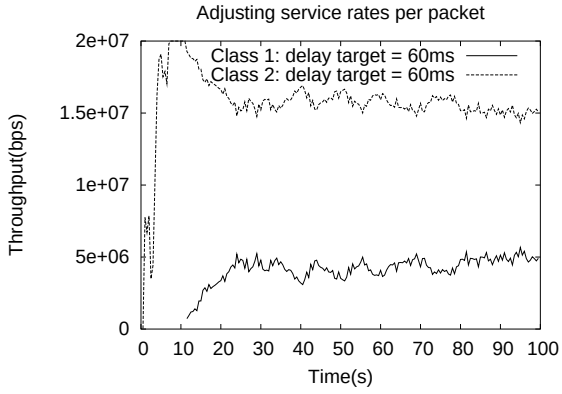


Figure 4.4: ICDS adjusts service rates in per-packet processing. The throughput of the two TCP classes does not converge to the ideal state: sharing the link bandwidth equally within 100 seconds.

with a small number of flows can be modelled and analyzed numerically. A system of non-linear equations are derived from several observations when the system is stable.

First, the sum of the service rates (i.e., the throughput) is the capacity of the bottleneck link C,

$$r_1 + r_2 = C, \tag{4.1}$$

where r_1 and r_2 are the service rate of TCP class 1 and TCP class 2 respectively.

Secondly, ICDS allocates the service rate r_i in proportion to the sum of the arrival rates (i.e., the sending rate s_i of class i) and a constant e (the add-rate option) for each class as computed by the following:

$$\frac{r_1}{r_2} = \frac{s_1 + e}{s_2 + e}. \tag{4.2}$$

Thirdly, the aggregate TCP congestion windows (the amount of data in transmission) is equal to the size of the network pipe (the sum of the router buffer size and the delay-bandwidth product), when a packet is dropped. At this time, all the TCP flows reach

the same maximal congestion window w_i because of the synchronization among a small number of TCP flows [6, 38] within a TCP class. Therefore, the following is true:

$$n_i w_i = (2T_p + d_i) r_i, \tag{4.3}$$

where n_i is the number of TCP flows. The delay-bandwidth product of class i is the product of the round trip propagation delay $2T_p$ and its service rate r_i. The buffer size of class i is the product of its delay target d_i and its service rate r_i.

Fourthly, the loss rate l_i of a TCP flow, which is also the loss rate of traffic class i because of the synchronization, is a function of the maximal congestion window w_i [38], represented by

$$l_i = \frac{8}{3w_i^2 + 21w_i + 8}. \tag{4.4}$$

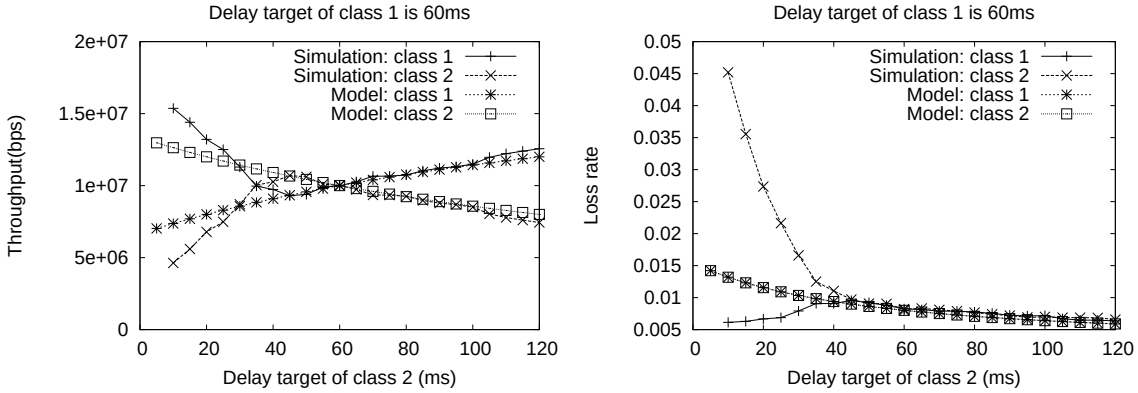
Finally, the following is true:

$$l_i = \frac{s_i - r_i}{s_i}. \tag{4.5}$$

The numerical values of s_i, r_i, and l_i can be calculated by solving the system of nonlinear equations with (4.1), (4.2), (4.3), (4.4) and (4.5) (by the GNU Scientific Library [1]).

A series of simulations with different parameters are conducted to verify the analysis. In each simulation, the data collected before the system is stable (100 seconds in the simulations) is discarded. Each point in Figure 4.5 is the average value of the throughput and the loss rate of one simulation between 100 and 200 seconds .

The first series of experiments study the impact of the delay targets on throughput and loss rates. The add-rate constant e is set at 0 to show the behaviour of the original ICDS without the add-rate option. The delay target of class 1 is fixed and set at 60ms (the two



(a) (b)

Figure 4.5: Behaviour of long TCP flows under the impact of delay targets.

way propagation delay); the delay target of class 2 is changed from 10ms to 120ms.

Figure 4.5 compares the simulation results with the numerical results of the analytical model. The simulation and analysis results match, if the delay target of class 2 is larger than 45 ms. Figure 4.5 shows that the class with a longer delay target receives less throughput, and the loss rates of the two traffic classes are equal. However, the model is not correct, when the delay target of class 2 is less than 45 ms. Class 2 is short of buffer, and its throughput decreases. The reason is that the assumption of Equation (4.4) no longer holds. Equation (4.4) is true only when the buffer is sufficient to guarantee the average congestion window size of TCP flows in terms of packets is larger than 3 [38].

The delay target of class 1 is 60 ms (the round trip propagation delay) in the previous experiments. Several similar experiments when the delay target of class 1 is less than or larger than 60 ms are conducted. The experiment results fall into two categories. (1) When the delay target of class 1 is larger than 45 ms, the experiment results are similar to the previous experiment results (the delay target of class 1 is 60 ms and the analysis model fits

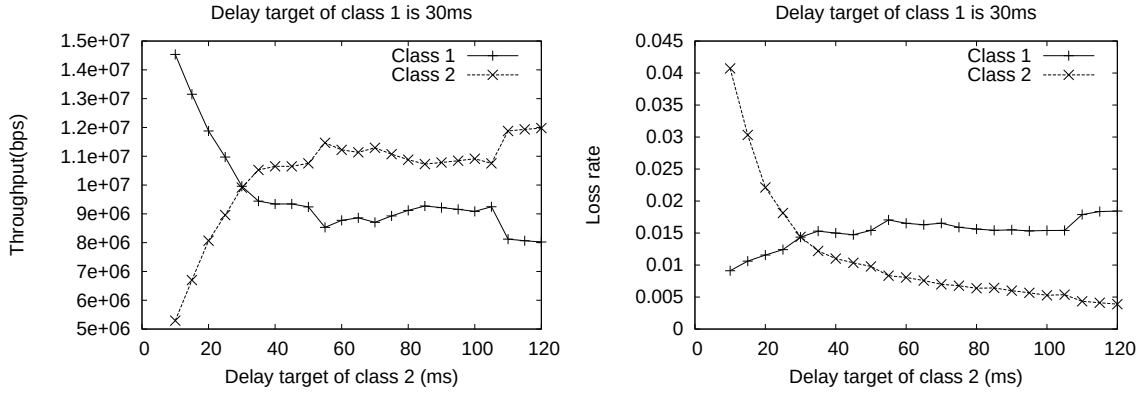


Figure 4.6: Behaviour of long TCP flows when the delay target of class 1 is 30ms.

with the simulation results in most areas). (2) When the delay target of class 1 is less than 45ms, the experiment results are different from the results when the delay target is 60ms. Figure 4.6 shows the results when the delay target of class 1 is 30 ms. The throughput of class 1 is smaller than class 2 and the loss rate of class 1 is larger than class 2, when the delay target of class 2 is larger than 30 ms although the queueing delay of class 1 is smaller than class 2. These observations violate the rule that TCP's throughput is inverse proportional to the round trip time. The reason is that the delay target of class 1 is so small that it is short of buffer. The throughput of class 1 is damaged and its loss rate is high. The analytical model does not fit the simulation results in this case (the analytical result is not shown in Figure 4.6) because the delay target of class 1 is so small that it lacks buffer and the assumption of Equation (4.4) does not hold.

Secondly, the impact of the add-rate constant e is studied. The delay target of class 1 and class 2 are set at 60 ms and 240 ms respectively. The add-rate constant e varies from 0 to 600kbps (transmitting 3MSS within the minimal delay target). Figure 4.7 shows the simulation results. When e increases, the difference between throughput decreases; the

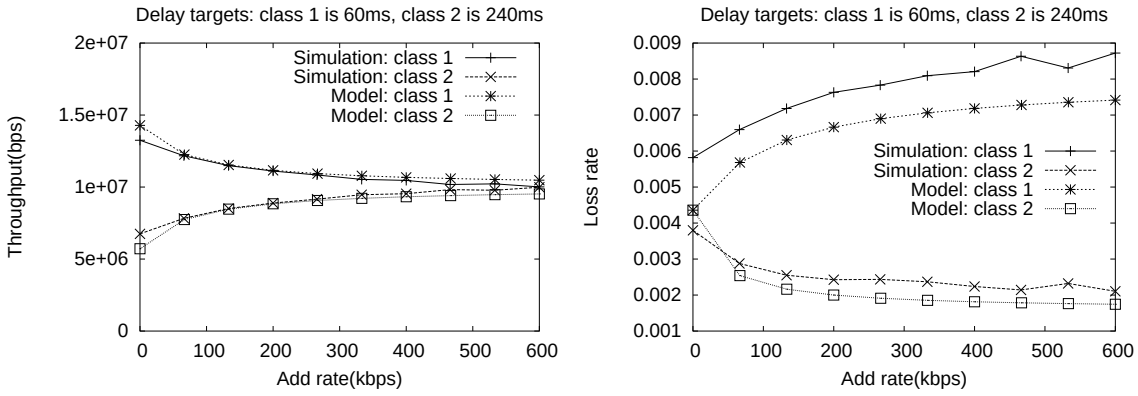


Figure 4.7: Behaviour of long TCP flows under the impact of the add-rate option.

loss rate of class 1 is larger than class 2, and the difference between them increases.

These observations can be explained as follows. The add-rate option e is identical in all the classes. The class with the low throughput r_1 benefits more from the add-rate option than the class with the high throughput r_2 because $\frac{e}{r_1} > \frac{e}{r_2}$, if $r_1 < r_2$. Therefore, the loss rate of the low-throughput class (class 2) decreases whereas the loss rate of the high-throughput class (class 1) increases; the throughput difference between them reduces.

Thirdly, the impact of the number of TCP flows in a class is investigated. Class 1 contains 10 TCP flows. The number of TCP flows in class 2 changes from 10 to 90. The delay targets of both classes are set at 60ms. Figure 4.8 shows the experiment results. The aggregate throughput of the class with more TCP flows is larger than the class with fewer TCP flows because the aggregate traffic of more flows is more aggressive. However, the loss rates of the two TCP classes are close even the number of flows in them are significant different. Therefore, in addition with the fact that the round trip time of the TCP flows in the two classes are identical, the throughput of one TCP flow in class 1 is the same as class 2 by TCP's throughput equation (Equation (2.3) on page 12); that is, ICDS allocates the

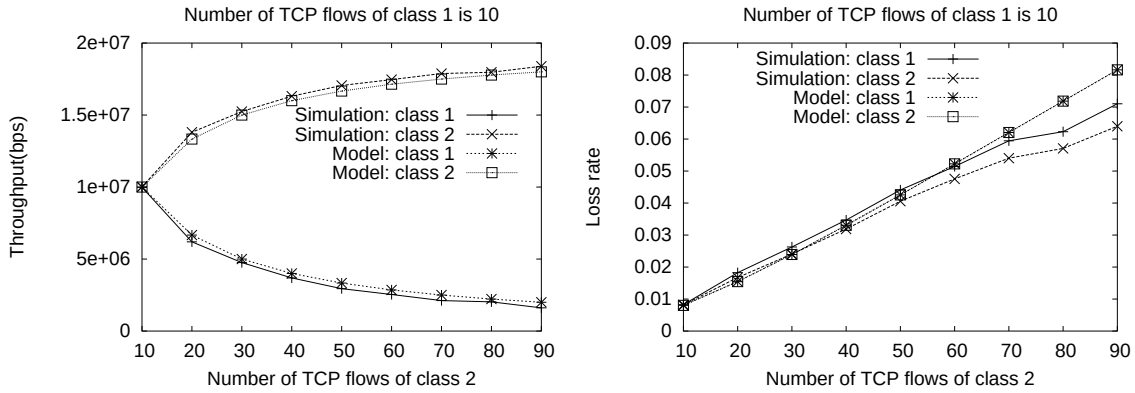


Figure 4.8: Behaviour of long TCP flows under the impact of the number of TCP flows.

amount of the service rate to a TCP class in proportion to the number of flows it contains. The loss rates of both classes increase when the total number of TCP flows in the system increases, because the total traffic is more aggressive.

Several experiments for a large number of desynchronized TCP flows are conducted. The experiment results are not shown here. The first observation is that the convergence of TCP traffic is fast even without the add-rate option. The possible reasons are the aggregate traffic is more aggressive with more number of flows and there is no synchronization effect. The numerical model for the small number of synchronized TCP flows does not match the simulation result of a large number of TCP flows because Equation (4.4), the foundation of the model, is derived under the assumption that TCP flows are synchronized and no longer holds. The loss rate of the simulation result is lower than the prediction of the model of a small number of TCP flows because the aggregate traffic of a large number of TCP flows is less bursty.

The analysis and the experiment results in this section demonstrate that ICDS preserves the properties of TCP traffic on DropTail queueing: the TCP flows with a smaller delay

target, therefore, a smaller round trip time, have higher throughput, and the TCP traffic with more flows is more aggressive. In other words, ICDS provides some degree of isolation for the TCP traffic of different classes. Furthermore, the analytical model is useful to understand and predict the performance characteristics of the TCP traffic in ICDS, given the value of the delay targets and the add-rate constant e.

After examining Figure 4.5 on page 61 again, it can be observed that when the delay target of class 2 is 45ms, the throughput of class 2 is at its maximum. If the delay target of class 2 is decreased to less than 45ms, the loss rate of class 2 increases dramatically. Clearly, the optimal delay target for the TCP flows in class 2 is 45ms.

4.5.2 Short Web-Like TCP Flows versus Long TCP Flows

Short web-like TCP flows arrive according to a Poisson process, and their file sizes comply to a Pareto distribution with an average of 30 packets [10] (the shape parameter of the Pareto distribution is 1.35 in the following simulations). This section studies the competition between a class with short web-like TCP flows and a class with long TCP flows.

Figure 4.9 shows that short web-like TCP flows use the fraction of link bandwidth corresponding to their aggregate load, while long TCP flows share the remaining bandwidth. The loss rate of short web-like TCP flows is higher than the long TCP flows because short web-like TCP flows stay in slow start [5] most of time, whereas long TCP flows always stay in congestion avoidance stage [5]. Slow start is more bursty than congestion avoidance.

Figure 4.9(b) shows that the loss rate of class 2 decreases slightly when its delay target is less than 30ms. This is a side effect of the minimal-rate mechanism. The minimal-rate mechanism sets a minimal rate for a class to guarantee that three packets can be transferred within the delay target (i.e., the minimal rate is calculated by 3MSS/ (the minimal delay target)); that is, if the delay target is smaller, the minimal rate is higher.

Consequently, the loss rate decreases if the traffic load is comparable to the minimal rate. In the following presentation, the data points affected by the side effect of the minimal-rate option are either not displayed in figures or are explained by simply referring to the side effect of the minimal-rate mechanism.

In addition, the experiments with the add-rate option enabled are conducted. Because the results are only slightly different, they are not shown here. A TCP class with a light load benefits more from the add-rate option than the one with a heavy load. Therefore, if the traffic load of the short web-like TCP class is 9.6% of the link bandwidth, its loss rate will decrease.

Figure 4.9 also shows the simulation result of DropTail queueing with the buffer size set at the delay-bandwidth product. The equivalent configuration in ICDS is that the delay targets of both classes are set at the round trip propagation delay (60ms). The throughput of the two classes are almost identical in ICDS and DropTail queueing. The loss rates are somewhat different. The reason is that ICDS provides some degree of isolation between long TCP flows and short web-like TCP flows, whereas DropTail queueing offers none. For example, in Figure 4.9(b), the loss rate of the short-web like TCP flows is significant higher than the long TCP flows in ICDS because ICDS cannot increase the service rate of the short TCP flows immediately if their arrival rate jumps up. However, with DropTail queueing, the long TCP flows are affected by the bursts of the short TCP flows immediately. Therefore, their loss rates are closer.

An interesting observation from Figure 4.9(b) to (d) is that the difference of loss rates between long TCP flows and short TCP flows with ICDS decreases. Furthermore, when the load of short TCP flows is 9.6%, the loss rate of short flows is higher than long flows in DropTail queueing, whereas their loss rates are similar, when the load of short TCP flows is 48%. The reason may be that low-load traffic tends to have higher loss rate than high-load

traffic. This seems true for both ICDS and DropTail queueing. Similar observations exist in the later simulations.

There is no clear point indicating the optimal delay target for short web-like TCP flows. The loss rate of short flows smoothly goes up when the delay target is decreased. One can argue that short web-like TCP flows must make a trade-off between a low loss rate and short queueing delay.

4.5.3 CBR UDP Traffic versus Long TCP Flows

In the following experiments, a CBR UDP class shares the bottleneck link bandwidth with a TCP class (containing 10 TCP flows). The delay target of the TCP class is set at 60 ms, and the delay target of the UDP class increases from 15 ms to 120 ms. The add-rate option is disabled to study the behaviour of the plain ICDS. Figure 4.10 shows that the CBR UDP class receives a share approximately equal to its sending rate (excluding the bandwidth wasted by the lost packets), whereas the TCP traffic occupies the remaining bandwidth. If the load of the UDP class is 20% of the bottleneck link, the loss rate of the UDP class increases when its delay target is less than 30ms. If the load of the UDP class is 80% of the bottleneck link, the loss rate of the UDP class stays low even when its delay target is small.

The different observations between the cases when the UDP load is 20% and 80% is due to the degree that the TCP traffic affects the CBR UDP traffic. Routers would not need buffers if only CBR UDP traffic exists. However, if it shares a bottleneck link with TCP traffic, its relative arrival rate $s_{UDP}/(s_{UDP} + s_{TCP})$ (s_{UDP} and S_{TCP} denote the absolute arrival rates of CBR UDP and TCP respectively) is also bursty; that is, although the absolute arrival rate of the CBR UDP traffic is constant, the service rate (equalling to the relative arrival rate) is bursty. How bursty the service rate of the CBR UDP traffic depends

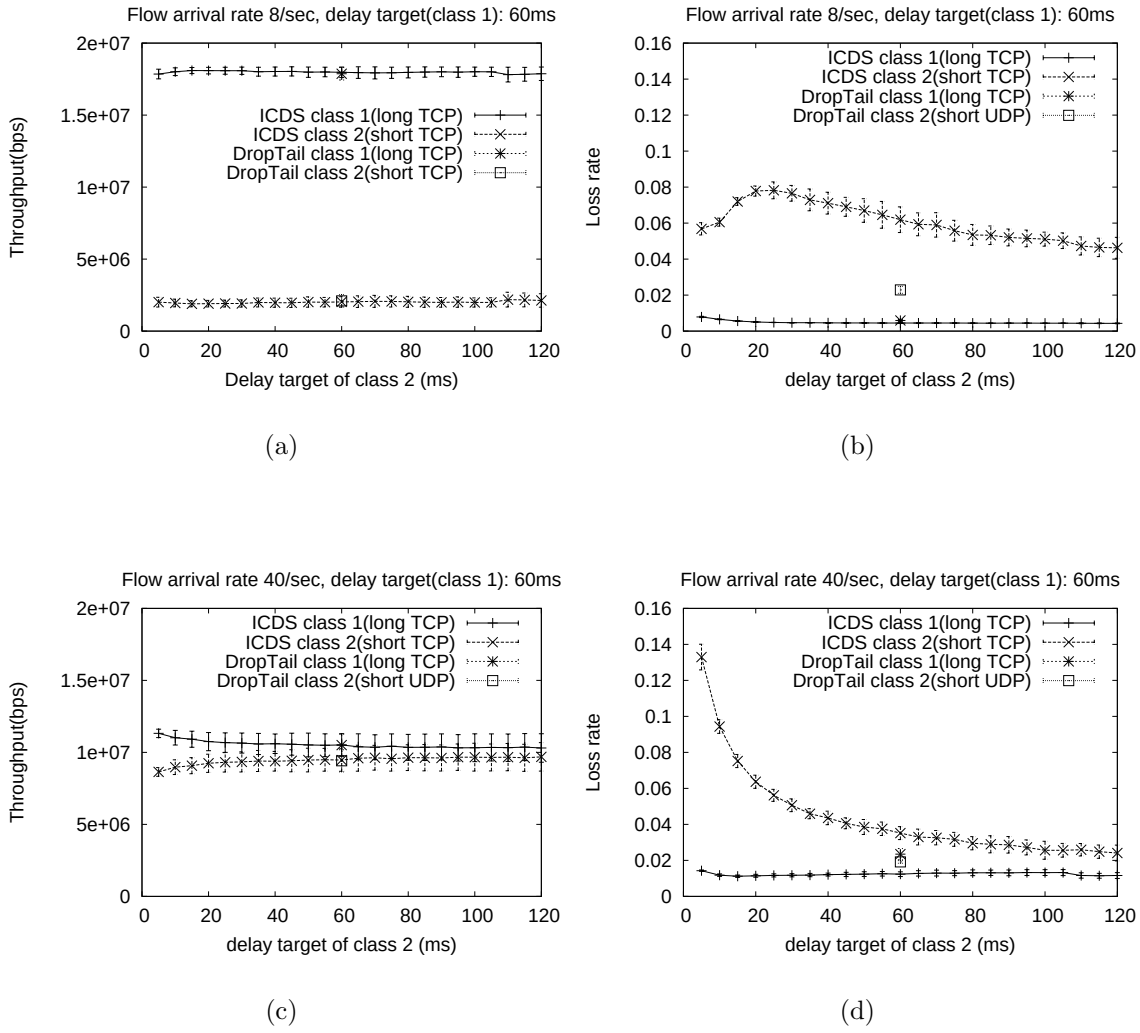


Figure 4.9: Short web-like TCP flows versus long TCP flows. The flow arrival rate of short web-like flows is 8/sec (i.e., the load is 9.6% of the link bandwidth) in (a) and (b). The flow arrival rate is 40/sec (i.e., the load is 48% of the link bandwidth) in (c) and (d).

on the fraction of its load in the total traffic. If the load S_{UDP} of CBR UDP is larger, its service rate is approaching 1. Therefore, the CBR UDP traffic has a near-constant service rate and less buffer is required. On the contrary, if the load of the CBR UDP traffic is small, its service rate is bursty and more buffer is needed.

If ICDS can isolate the CBR UDP traffic completely from the TCP traffic, then the delay target of the CBR UDP class can be arbitrarily small. The experiment results in Figure 4.10(b)) shows that ICDS cannot isolate different traffic completely because traffic classes share the link bandwidth.

Figure 4.10 also shows the throughput and the loss rates of the TCP class and the CBR UDP class in DropTail queueing. The throughput of two classes in ICDS is identical with that of DropTail queueing. However, the loss rates are different. The reason is that ICDS uses ETSW, a smoothing algorithm, to smooth the estimation of arrival rates. Because the TSW window size is tuned for TCP traffic, the bursts of TCP traffic is effectively removed. Therefore, the difference of the loss rates between the CBR UDP traffic and the TCP traffic is small. For DropTail queueing, there is no such smoothing such that the loss rate difference between the two classes is larger.

Also, the simulation with the add-rate option is conducted but not shown here. When the load of the CBR UDP traffic is 20% of the link capacity, the add-rate option plus the minimal-rate mechanism is comparable to the load of the CBR UDP traffic. The result is that the CBR UDP traffic does not lose packets any more. When the load of the CBR UDP traffic is 80% of the link capacity, the loss rate of the CBR UDP traffic is higher than the TCP traffic, because the add-rate option has no impact on the non-responsive UDP flows, whereas the TCP traffic benefit from the add-rate option.

When the load of UDP traffic is 20% of the link capacity, the optimal delay target seems at the 50ms, whereas when the load of UDP traffic is 80% of the link capacity, the

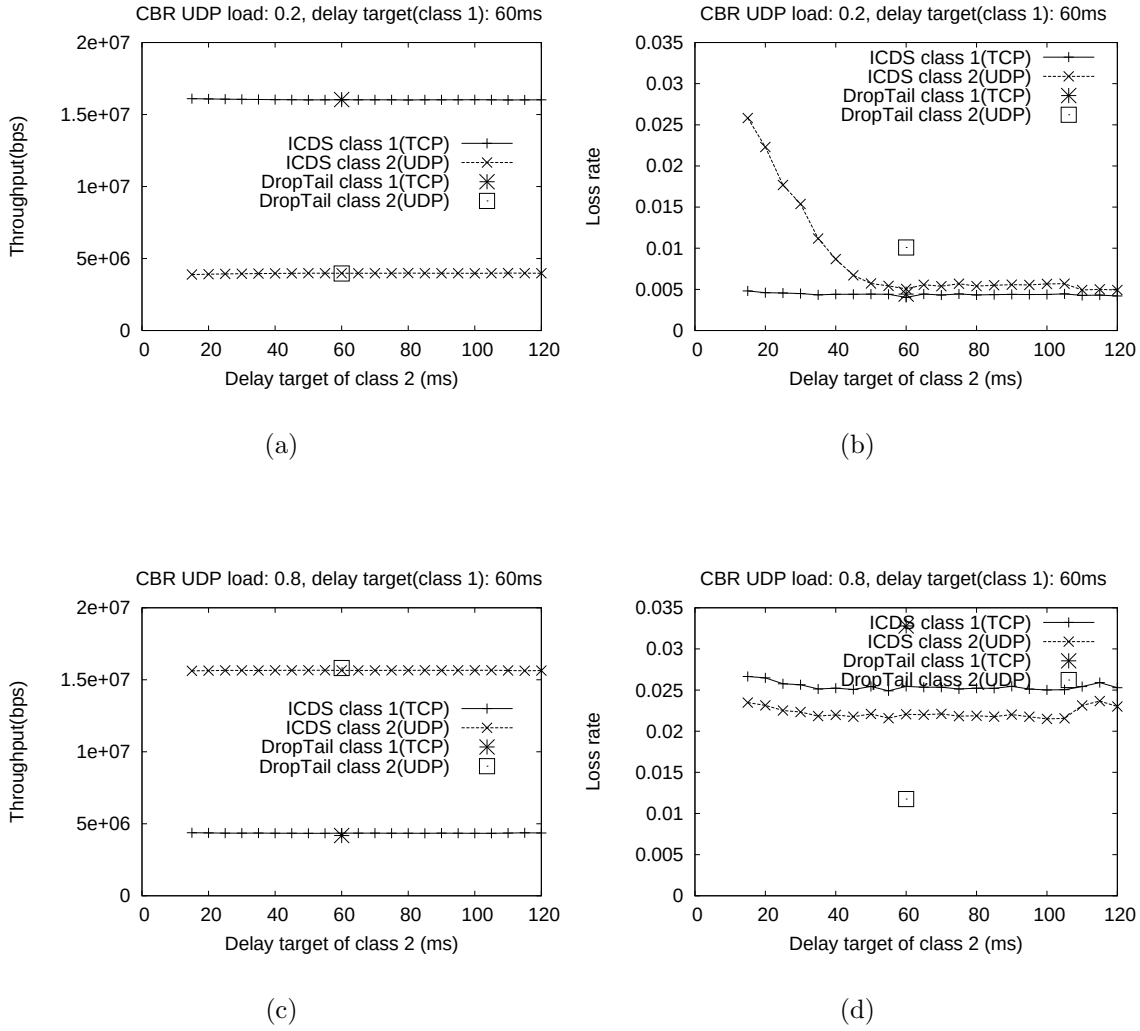


Figure 4.10: CBR UDP traffic versus long TCP flows.

optimal delay target is less than 20ms.

4.5.4 Self-Similar UDP Traffic versus Long TCP flows

This section studies the behaviour of ICDS when self-similar UDP traffic shares link bandwidth with TCP traffic. The self-similar traffic is generated by ON/OFF heavy-tail sources [43]. 32 Pareto sources [25] (a typical self-similar generator) with the shape parameter 1.4 are used in these experiments. The add-rate option is disabled to show the behaviour of plain ICDS. Figure 4.11 shows the experiment result. Self-similar UDP traffic is much more bursty than TCP traffic. Therefore, the self-similar UDP traffic's loss rate is much higher than the TCP traffic no matter whether its traffic load is high or low. The loss rate of class 2 decreases, when the delay target is less than 25ms in Figure 4.11(b) because of the side effect of the minimal-rate mechanism.

Figure 4.11 also shows the experiment result of DropTail queueing. The loss rates of the self-similar UDP traffic is very close to the TCP traffic in DropTail queueing, whereas they are significantly different in ICDS. The throughput in ICDS are also different from DropTail Queueing. The reason is that ICDS provides some isolation to protect the TCP traffic from the bursts of the self-similar traffic, whereas DropTail queueing does not isolate them at all. When the arrival rates of the self-similar traffic increases significantly in a very short time period, ICDS cannot adjusts the service rate of the self-similar traffic immediately because of the smoothing component in the rate estimator and the control delay. Therefore, the self-similar traffic looses packets whereas the TCP traffic is not affected. On the contrary, the increase of the self-similar traffic affects the TCP traffic immediately in DropTail queueing. Consequently, their loss rates are similar.

The simulation results of self-similar UDP traffic competing with TCP long flows in ICDS with the add-rate option enabled are almost the same as the previous results with

the add-rate disabled. They are not shown in here.

It seems that there is no clear indication on where the optimal delay target for the self-similar traffic. If the load of self-similar traffic is 20% of the link capacity, the optimal delay target can be very small because the loss rate is always high. The self-similar traffic has to choose a delay target as a trade-off between a low loss rate and a short queueing delay, when the load of the self-similar traffic is 80% of the link capacity.

4.5.5 TFRC Flows versus Long TCP flows

This section examines the behaviour of ICDS, when TFRC traffic shares a bottleneck link with TCP traffic. The number of the TCP flows and the TFRC flows are both 10 in the experiments. The delay target of the TFRC class increases from 8 ms to 120 ms. Figure 4.12 shows that the throughput of the TFRC traffic is slightly higher than the TCP traffic when their delay targets are both 60ms. The loss rate of the TCP traffic is slightly higher than the TFRC traffic when the delay target of class 2 changes from 14ms to 60ms. Furthermore, the required buffer size of the TFRC traffic is less than the TCP traffic as inferred by comparing the ranges of delay targets at which the loss rates of them increase dramatically. The loss rate of the TFRC traffic increases dramatically when its delay target is less than 14ms from Figure 4.12(b). However, the loss rate of TCP traffic increases dramatically when its delay target is less than 45ms from Figure 4.5(b). All these observations indicate that the TFRC flows have a slight advantage (a lower loss rate and higher throughput) than long TCP flows in the same simulation configuration. This occurs because TFRC traffic is smoother than TCP traffic although they have the same response function [20].

Figure 4.12 also shows that the throughput and the loss rates of the TFRC traffic and TCP traffic on DropTail queueing are almost the same with ICDS. Similarly, the

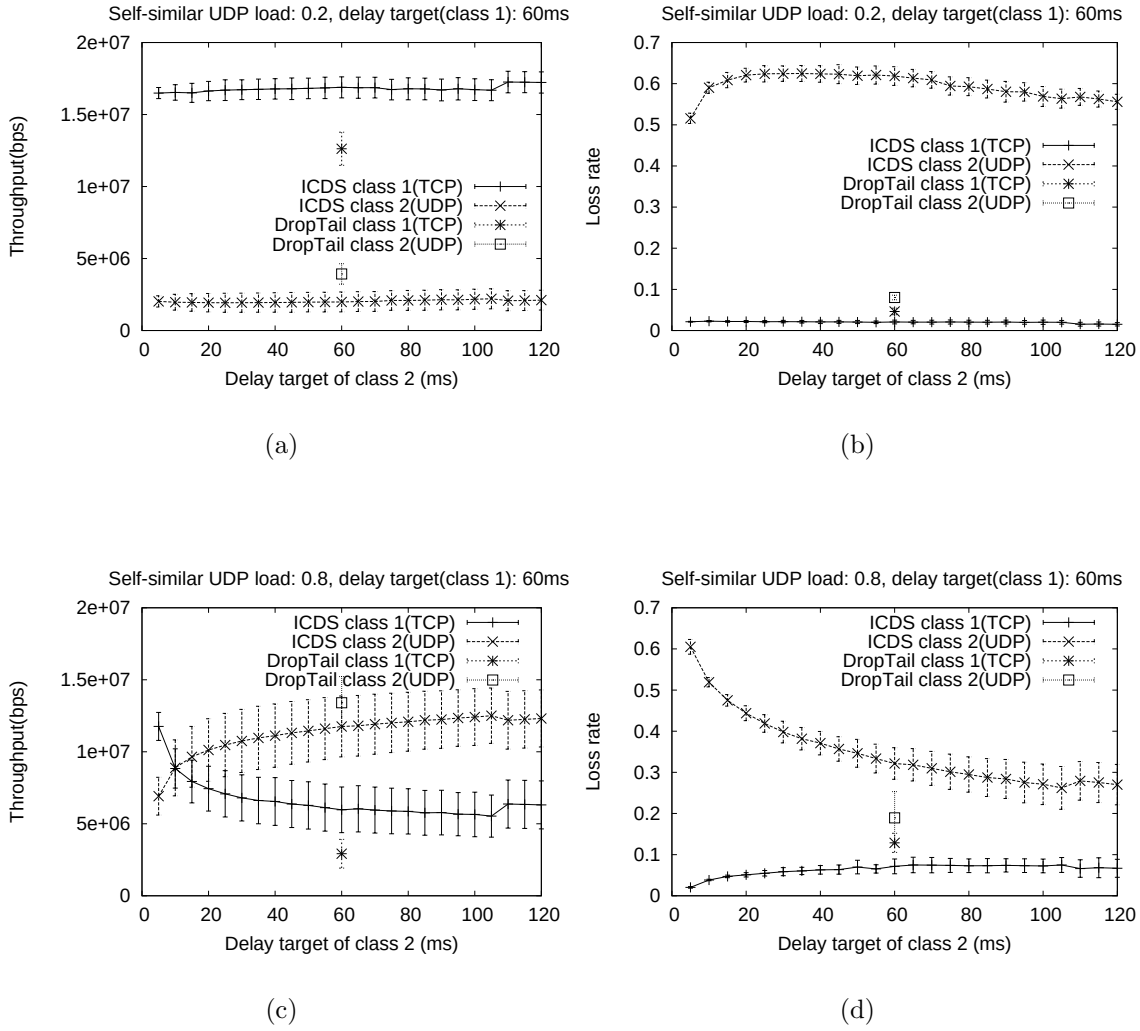
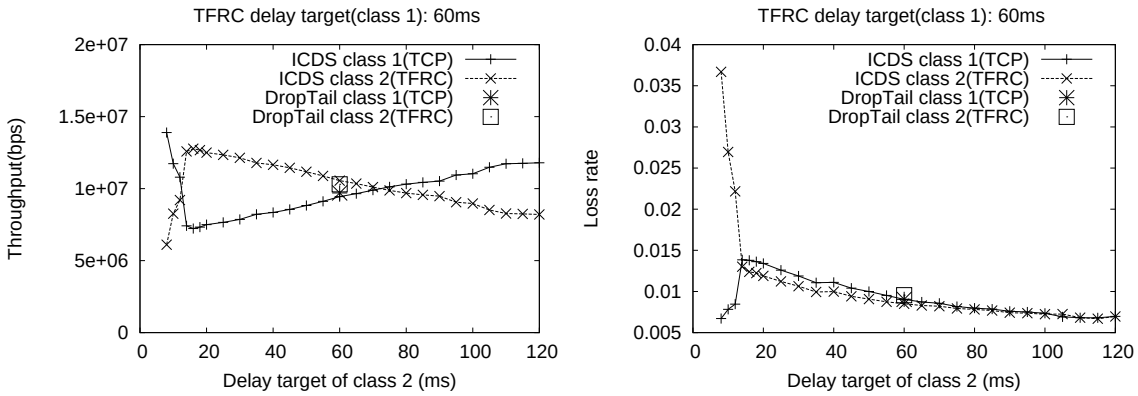


Figure 4.11: Self-similar UDP traffic versus long TCP flows.



(a) (b)

Figure 4.12: TFRC flows versus long TCP flows.

throughput of the TFRC traffic is slightly higher than the TCP traffic and the loss rate of the TFRC traffic is slightly lower than the TCP traffic because TFRC traffic is smoother than TCP traffic.

For TFRC traffic competing with TCP traffic, the add-rate option has an impact similar to the case when long TCP flows compete with long TCP flows as shown in Figure 4.7 on page 63; that is, the throughput difference is reduced and the loss-rate difference increases. The loss rate of the low-throughput traffic is less than the loss rate of the high-throughput traffic because the low-throughput traffic benefits more from the add-rate option.

The experiment results in this section show that ICDS preserves the characteristics of TFRC traffic and TCP traffic. The optimal delay target of TFRC flows exists at 16ms from Figure 4.12.

4.5.6 Optimal Delay Targets

Most of the previous experiments in this chapter change the delay target of the class competing with long TCP flows. The simulation results indicate that an *optimal* delay target usually exists. If the delay target of a class is less than the optimal delay target, its loss rate increases significantly. Moreover, these simulation results reveal that this optimal delay target are usually related to the burst characteristics of this traffic. Smooth traffic has a small optimal delay target, whereas bursty traffic has a large optimal delay target. For example, the optimal delay target of TFRC flows is less than the optimal delay target of TCP flows.

The experiments in this section are an attempt to answer questions from the opposite perspective. When one special type of traffic (long TCP flows) competes with other different types of traffic, does an optimal delay target exist for this special traffic? How is this optimal delay target related to the burst characteristics of this traffic and the traffic it competes with?

Figure 4.5 on page 61 shows the experiment results when long TCP flows competes with long TCP flows. Figures 4.13, 4.14, 4.15, and 4.16 show the results when long TCP flows competes with other different types of traffic. These experiments change the delay target of the long TCP flows.

These simulations demonstrate that when decreasing the delay target of long TCP flows, the loss rate of TCP flows sometime increases as shown in Figure 4.14(b), sometime keeps near constant and the loss rate of the other traffic decreases as shown in Figure 4.15(b). These observations can be explained by the degree of bursts between long TCP flows and the other type of traffic. If the TCP traffic is more bursty than the other traffic (CBR UDP traffic in Figure 4.14(b)), decreasing the delay target of TCP traffic increases its loss rate. If TCP is less bursty than the other traffic, decreasing the delay target of the

TCP traffic is detrimental to it but the effect is exhibited at the more bursty traffic: the loss rate of the bursty traffic decreases because it benefits when TCP traffic is damaged by short of buffer.

When the delay target of the TCP traffic decreases, the loss rate of the long TCP flows does not increase dramatically if the other traffic is more bursty. A new metric is used to define the *optimal* delay target. If the delay target is less than the optimal delay target, the throughput decreases. Most of the simulation results show that an optimal delay target for the long TCP flows exist with two exceptions. (1) Figure 4.14(c) shows that the throughput of the TCP traffic keeps constant when its delay targets is decreased. However, Figure 4.14(d) shows that the loss rate of the TCP traffic is higher than the UDP traffic, when its delay target is less than 25ms. It seems reasonable to consider this value as the optimal delay target. (2) Figure 4.15(a) exhibits that the throughput of the TCP traffic degrades, when its delay target decreases in the whole range between 5ms and 120ms. There is no clear turning point on the curve for the optimal delay target. It is argued that in such scenario, the TCP class must choose an "optimal" delay target as a trade-off between high throughput with a long queueing delay and low throughput with a short queueing delay.

These simulation results do not clearly indicate where the optimal delay target exists for the TCP traffic, but it is clear that the optimal delay target is not only determined by the burst characteristic of the TCP traffic but also the characteristic of the traffic with which the TCP traffic competes.

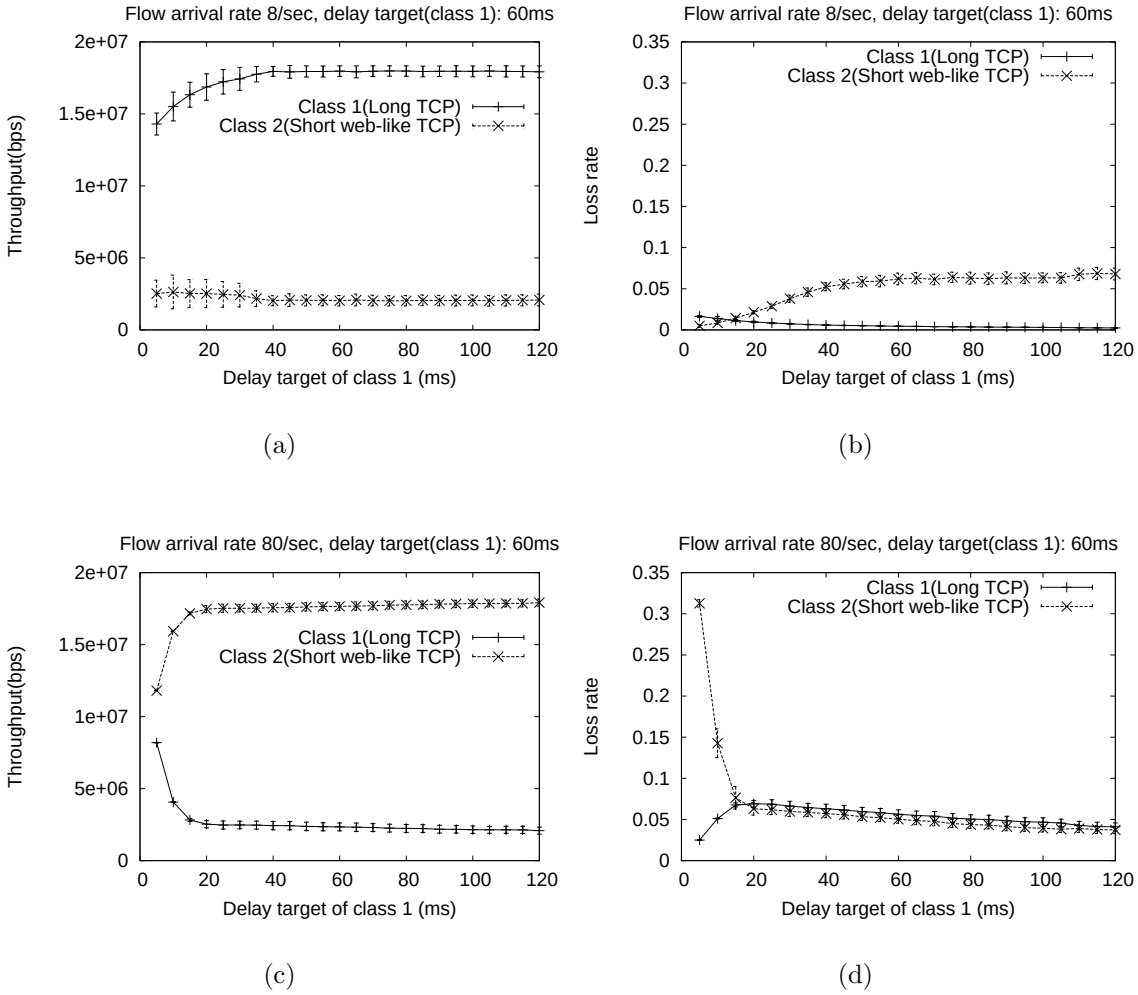


Figure 4.13: Long TCP flows versus short web-like TCP flows. The delay target of class 1 (traffic of long TCP flows) increases from 5 ms to 120 ms.

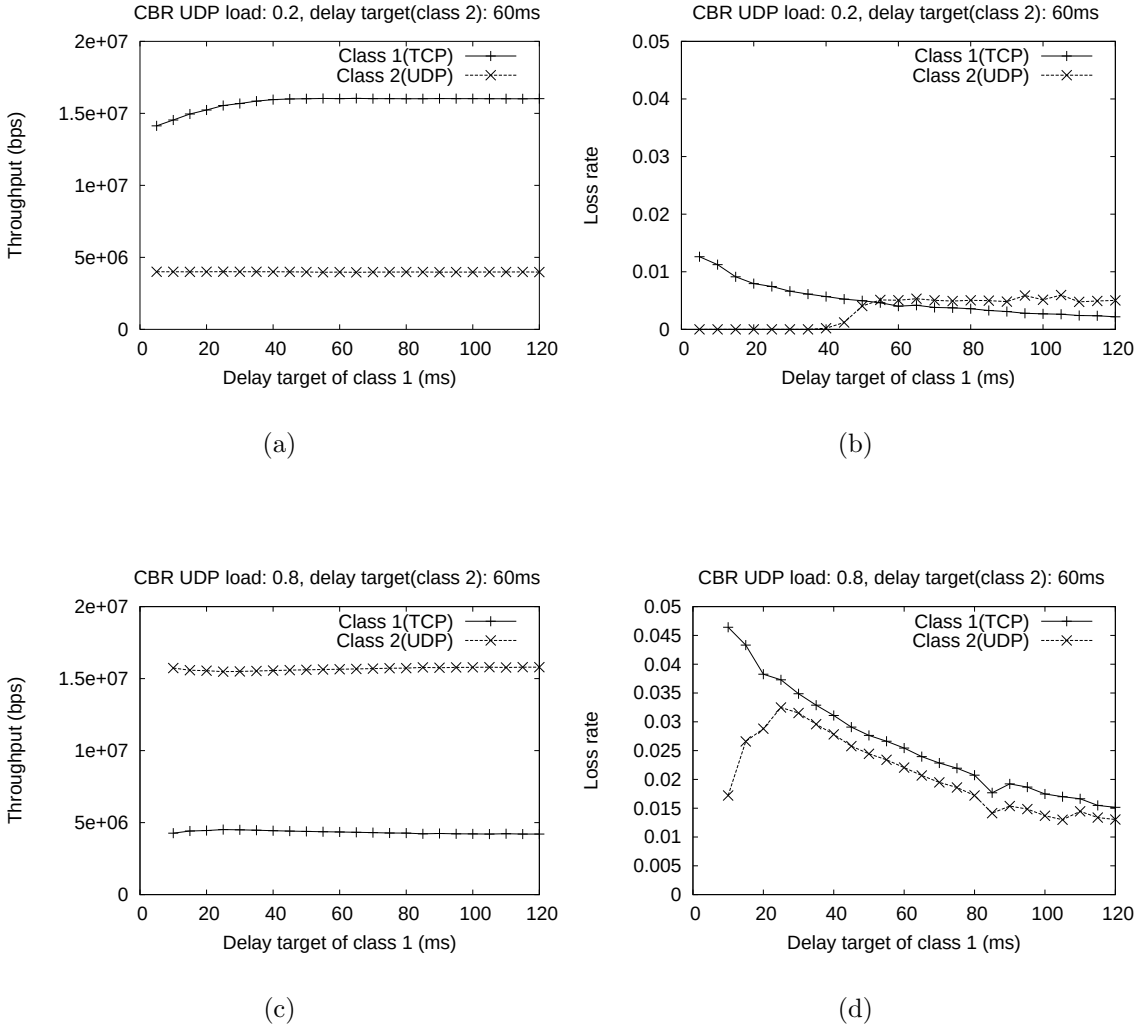


Figure 4.14: Long TCP flows versus CBR UDP Traffic. The delay target of TCP traffic increases from 10 ms to 120 ms.

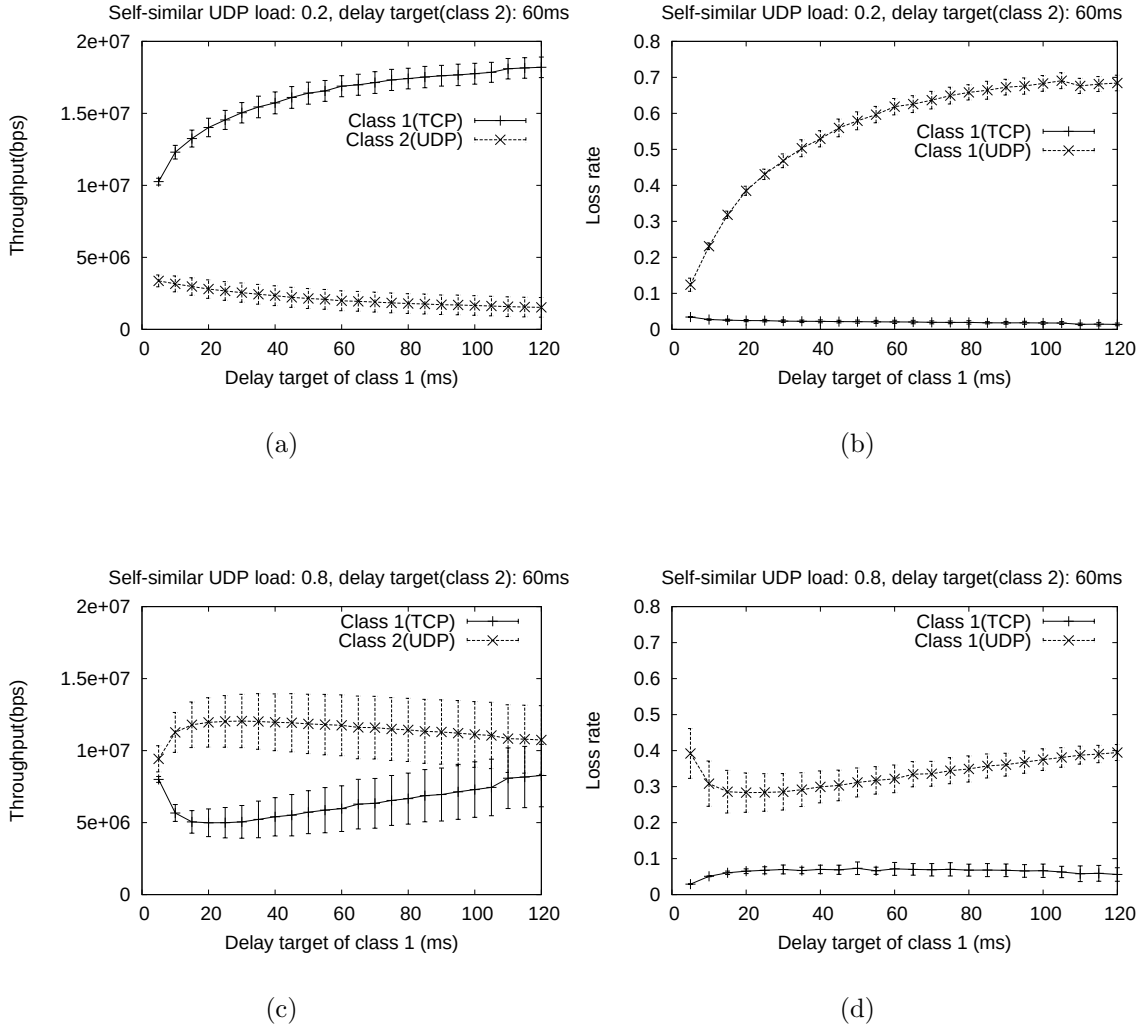
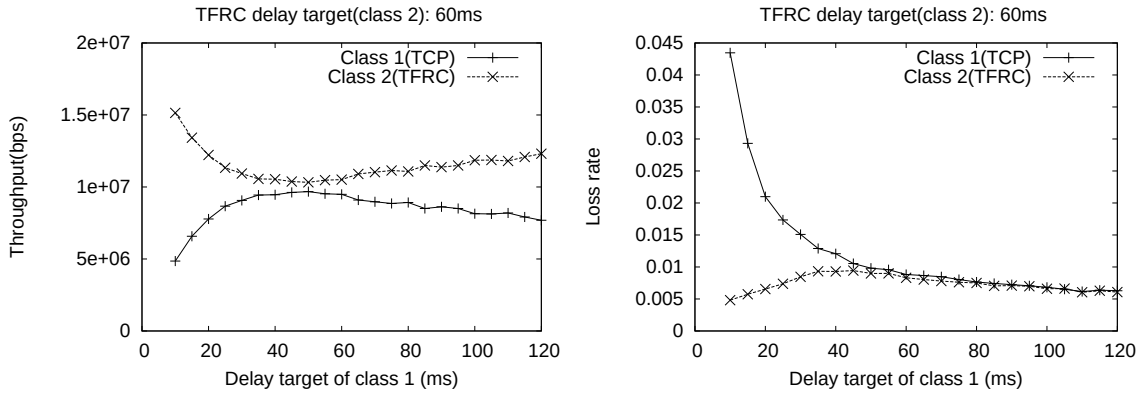


Figure 4.15: Long TCP flows versus self-similar UDP traffic. The delay target of TCP traffic increases from 5 ms to 120 ms.



(a) (b)

Figure 4.16: Long TCP flows versus TFRC flows. The delay target of TCP traffic increases from 10 ms to 120 ms.

Chapter 5

Conclusions and Future Work

5.1 Conclusion

This thesis presents the implementation and the evaluation of *Incentive-compatible Differentiated Scheduling* (ICDS). The evaluation of ICDS for different types of traffic demonstrates that ICDS can mostly isolate and preserve the characteristics of different traffic classes. The results of the simulation and analysis are summarized as follows:

1. The minimal-rate mechanism is critical for TCP traffic to start transmitting data.

2. The add-rate option can boost the convergence of a small number of synchronized TCP flows. A TCP class with low throughput benefits more from the add-rate option than a TCP class with higher throughput because the value of the add-rate option is identical for all the classes. However, the add-rate option has no impact on non-responsive UDP flows. Furthermore, desynchronized TCP flows converge fast even without the add-rate option.

3. The appropriate TSW window size with fast convergence and no oscillation is the

average round trip propagation delay for TCP traffic.

4. ICDS must adjust the service rates of different classes in the same frequency to provide the same chance to increase service rate for the traffic with different rates. This is important for the convergence of TCP traffic.

5. This thesis provides a numerical model for long TCP flows. This model verified by simulations indicates that ICDS preserve the behaviour of TCP traffic. The long TCP flows in a class with a smaller delay target has higher throughput than a class with a larger delay target. ICDS allocates service rates in proportion to the number of flows in each class. This numerical model is useful to predict the throughput and the loss rate of a class given the delay target and the number of flows.

6. If the background traffic is the same, a bursty traffic class needs more buffer than a smooth traffic class does.

7. In most cases, an optimal delay target exists for a certain type of traffic. However, the value of this optimal delay target is related not only to the characteristic of this traffic, but also to the characteristic of the background traffic.

8. ICDS provides some isolation between different types of traffic. For example, ICDS protects TCP traffic from the extremely bursty self-similar UDP traffic. The reason is that ICDS smooths the estimated arrival rate and a time lag exists between the arrival-rate estimation and the service-rate adjustment. Consequently, if the extremely bursty traffic increases its arrival rate dramatically, ICDS cannot ajust the service rate so quickly such that the bursty traffic loses packets. On the contrary, DropTail queueing does not offer such protections because it serves smooth traffic and burst traffic together. Smooth traffic is affected if bursty traffic increases its

sending rate dramatically.

The evidences from the simulation and analysis demonstrate that it is likely that ICDS can satisfy its design goal. Applications can choose the service classes according to the burst characteristics of their traffic. ICDS does not require admission control and charging mechanisms. Consequently, it does not have the high implementation overhead of control plane mechanisms and can be deployed incrementally.

5.2 Future Work

As illustrated in Section 3.3, to provide strict delay guarantee to traffic classes, ICDS loses some link utilization. In addition, many applications, such as the interactive web applications, are not very sensitive to instantaneous increase of packet delay as long as the majority of the packet delays are within the delay target. A future variant of ICDS is to provide service classes with loose delay guarantee and increase the link utilization.

The numerical model for TCP traffic with ICDS in thesis is applicable for only a small number of synchronized long TCP flows. It would be interesting to exploit the model to describe the behaviour of a large number of desynchronized long TCP flows in ICDS. Also, the numerical model cannot depict the behaviour of TCP traffic, when the delay target is very small. The buffer of this class is not large enough to provide full utilization of the allocated service rate or fairness among the TCP flows. Clearly, a model manifests the case of the small delay target is desirable.

The current simulations are all conducted on the standard dumbbell topology. The simulations on other type of topologies, such as multihop topology, are planed to investigate the behaviour of ICDS in different scenarios.

In addition, ICDS can be used as a per-hop service in the architecture of DiffServ.

Such systems provide end-to-end delay guarantee. Consider a network composed by ICDS routers with admission control employed on the gateways at the edge. If the total load of all the input traffic is guaranteed to be less than the capacity of the interval network by the admission control, the total end-to-end queueing delay may be the delay target of the service class that is chosen for the applications, independent of the number of hops of the data path. This is similar to how a network composed of GPS routers provides guaranteed delay services to the traffic shaped by token buckets. The difference is that the GPS approach requires end-to-end signalling and rate reservation, whereas the ICDS approach requires admission control only at the edge router with automatic rate allocation; that is, for ICDS, the service rates are adjusted in proportion to the arrival rates of the traffic classes). This application of ICDS requires further investigation.

Finally, a variant of ICDS can similarly be used for other non-elevated services in which delay is traded for throughput. A throughput penalty, such as allocating the service rate at 90% of the relative arrival rate, can be applied to the short-delay classes. By tuning the parameter of the penalty, the short-delay class has lower throughput and a higher loss rate, whereas the long-delay class has higher throughput and a lower loss rate.

5.3 Contributions

The basic idea of ICDS is conceived by Dr. Martin Karsten. The contributions of this thesis are restated as follows.

1. This thesis investigates the behaviour of ICDS by analysis and simulations under different types of traffic. In particular, it provides a model to describe the behaviour of TCP traffic with ICDS based on an existing model on the behaviour of TCP traffic with DropTail queueing. It shows some evidences that ICDS can achieve its design

goal.

2. The examination of the convergence of TCP traffic in ICDS leads to a clear understanding of the effects of the minimal-rate mechanism and the add-rate option.

3. The investigation in this thesis discovers that it is important to adjust service rates at the same frequency for all the classes in ICDS to achieve a fast convergence for TCP traffic.

4. This thesis also provides numerical analysis for the bounds of the errors introduced by the technique to remove division operations, and a simple proof of the long-term decay property of the *Efficient Time Sliding Window* (ETSW) algorithm.

Bibliography

[1] GNU Scientific Library. http://www.gnu.org/software/gsl/.

[2] Maple. http://www.maplesoft.com/.

[3] Network Simulator (ns-2). http://www.isi.edu/nsnam/ns/.

[4] QBone Premium Service. http://qbone.internet2.edu/premium/.

[5] Mark Allman, Vern Paxson, and W. Richard Stevens. TCP congestion control. *IETF RFC 2581*, April 1999.

[6] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. In *ACM SIGCOMM'04*, Portland, Oregon, September 2004.

[7] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated service. *IETF RFC 2475*, December 1998.

[8] R. Braden, D. Clark, and S. Shenker. Integrated services in the Internet architecture: an overview. *IETF RFC 1633*, June 1994.

[9] David D. Clark and Wenjia Fang. Explicit allocation of best effort packet delivery service. *IEEE/ACM Transactions on Networking*, 6(4), August 1998.

[10] Mark E. Crovella and Azer Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, December 1997.

[11] B. Davie, A. Charny, J.C.R. Bennett, K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. An Expedited Forwarding PHB (Per-Hop Behavior). *IETF RFC 3246*, March 2002.

[12] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and simulation of a fair queueing algorithm. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 1–12, Austin, Texas, September 1989.

[13] Constantinos Dovrolis and Parameswaran Ramanathann. Proportional differentiated services, part II: Loss rate differentiation and packet dropping. In *Proc. International Workshop on Quality of Service*, June 2000.

[14] Constantinos Dovrolis, Dimitrios Stiliadis, and Parameswaran Ramanathan. Proportional differentiated services: delay differentiation and packet scheduling. In *SIGCOMM*, pages 109–120, 1999.

[15] Wenjia Fang. The "Expected Capacity" famework: simulation results. *Princeton University Computer Science Technical Reports, TR-601-99*, March 1999.

[16] Victor Firoiu and Marty Borden. A study of active queue management for congestion control. In *IEEE INFOCOM*, 2000.

[17] Victor Firoiu, Xiaohui Zhang, and Yang Guo. Best effort differentiated services: Trade-off service differentiation for elastic applications. In *IEEE ICT*, June 2001.

[18] Sally Floyd. Questions about NewReno deployment. http://www.icir.org/floyd/newreno-questions.html, October 2000.

[19] Sally Floyd. Questions about SACK Deployment. http://www.icir.org/floyd/sack-questions.html, October 2000.

[20] Sally Floyd, Mark Handley, Jitendra Padhye, and Jorg Widmer. Equation-based congestion control for unicast applications. In *ACM SIGCOMM'2000*, 2000.

[21] Sally Floyd and Van Jacobson. On traffic phase effects in packet-switched gateways. *Journal of Internetworking:Practice and Experience*, 3(3):115–156, September 1992.

[22] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4), August 1997.

[23] Chuck Fraleigh. *Provisioning Internet Backbone Networks to Support Latency Sensitive Applications.* PhD thesis, Stanford University, Department of Electrical Engineering, June 2002.

[24] Benjamin Gaidioz and Pascale Primet. EDS: a new scalable service differentiation architecture for internet. In *IEEE ISCC'02*, 2002.

[25] Glen Kramer. Generator of self-similar network traffic (version 2). http://wwwcsif.cs.ucdavis.edu/ kramer/code/trf_gen2.html.

[26] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. *IETF RFC 2597*, June 1999.

[27] Janey C. Hoe. Improving the start-up behavior of a congestion scheme for TCP. In *SIGCOMM'96*, August 1996.

[28] Paul Hurley, Patrick Thiran, Mourad Kara, and Jean-Yves Le Boudec. ABE: Providing a low-delay service within best effort. *IEEE Network Magazine*, 15(3), May 2001.

[29] Van Jacobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM'88*, August 1988.

[30] Van Jacobson. Modified TCP congestion avoidance algorithm. *email to the end2end list*, April 1990.

[31] Leonard Kleinrock. *Queueing Systems Vol 2: Computer Applications*. Wiley, New York, 1976.

[32] Thomas J. Kostas, Michael S. Borella, Ikhlaq Sidhu, Guido M. Schuster, Jacek Grabiec, and Jerry Mahler. Real-time voice over packet-switched networks. *IEEE Network*, (18–27), January 1998.

[33] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgment options. *RFC 2018*, October 1996.

[34] Robert Morris. TCP behavior with many flows. In *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, pages 205–211, October 1997.

[35] Wael Noureddine and Fouad Tobagi. The Transmission Control Protocol. http://mmnetworks.stanford.edu/waelnour/docs/TCP-doc.ps, July 2002.

[36] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *SIGCOMM '98*, pages 303–314, 1998.

[37] Abhay K. Pareh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transaction on Networking*, 1(3):344–357, June 1993.

[38] Lili Qiu, Yin Zhang, and Srinivasan Keshav. Understanding the performance of many TCP flows. *Computer Networks*, 37(3-4):277–306, 2001.

[39] P. Soderquist and M. Leeser. Analysis of the impact of different methods for division/square root computation in the performance of a superscalar microprocessor. *IEEE Micro*, 17(4):56–66, July 1997.

[40] Ben Teitelbaum and Stanislav Shalunov. Why premium IP service has not deployed (and probably never will). *Internet2 QoS Working Group Informational Document*, May 2002.

[41] Paolo Valente. Exact GPS simulation with logarithmic complexity, and its application to an optimally fair scheduler. In *SIGCOMM'04*, 2004.

[42] Curtis Villamizar and Cheng Song. High performance TCP in ANSNET. *ACM SIGCOMM Computer Communication Review*, 24(5), October 1994.

[43] Walter Willinger, Vern Paxson, and Murad S. Taqqu. Self-similarity and heavy tails: structural modeling of network traffic. *A practical guide to heavy tails: statistical techniques and applications*, pages 27–53, 1998.

[44] Hui Zhang. Service disciplines for guaranteed performance service in packet-switching network. *Proc. IEEE*, 83(10), October 1995.